Optimization of Automata for Natural Language Dependency Parsing

Martín Ariel Domínguez

Grupo de Procesamiento de Lenguaje Natural
Facultad de Matemática, Astronomía y Física
UNIVERSIDAD NACIONAL DE CÓRDOBA

Presentado ante la Facultad de Matemática, Astronomía y Física como parte de los requisitos mínimos para la obtención del grado de *Doctor en Ciencias de la Computación.*

junio, 2012

Director: Gabriel Infante-Lopez

Abstract

Natural languages phenomena have been studied by linguistic tradition well before the invention of computers. When computers appeared and large quantities of data could be processed a new, more empirical, approach to the study of languages arose. Nowadays we can test and derive hypotheses from the automatic processing of a huge amount of digitalized texts.

This thesis is concerned with different ways of benefiting from the technological possibility to refine and justify former knowledge based on more rationalistic treatments of Syntax in Natural Language Processing (NLP). We present different approaches where we apply computational methods to NLP. In two of them, we employ Genetic Algorithms to automatically infer data driven solutions to problems which were treated manually in previous works. Namely, the construction of Part-of-Speech tag sets and the finding of heads of syntactic constituents. In the third approach, we go a step further and propose an architecture for building multi-language unsupervised parsers that can learn structures based just on samples of data.

We use the formalism of Bilexical Grammars as a way to model syntactical structures of sentences throughout our whole thesis. As Bilexical Grammars are based on finite automata, we inherit their good learnability properties. In addition, our experiments with these grammars gave us practical understanding about their properties.

From this three works we see that by using automata we can model syntactic structures for different purposes. The results we obtain with our three different approaches look promising. In the first two approaches concerning supervised phrase structure parsing, we improve the performance of two state-of-the-art parsers. In our third work, we obtain state-of-the-art results with our unsupervised dependency parser for eight different languages.

keywords: natural language processing, dependency trees, phrase structure trees, syntactic parsing, supervised phrase structure parsing, unsupervised dependency parsing, bilexical grammar, grammar induction, automata induction, genetic algorithm, EM algorithm.

Classification:

- I.2.7 Natural Language Processing: Language models, Language parsing and understanding.
- I.2.6 Learning: induction.
- G.1.6 Optimization.
- F.4.2 Grammars and Other Rewriting Systems .
- G.3 Probability and statistics.

To my wife Victoria, my daughter Julieta and my parents Alfredo and Maruca.

Acknowledgements

This thesis would not have been possible without the support of many people. First and foremost I would like to thank my wife, Victoria, who in addition to being the person I chose for sharing my life and dreams, she helped me make this thesis ”readable” in English.

I would like to acknowledge the debt I owe to my parents, Alfredo and Maruca, because they are a model to me and beside their scarce academic training they have always encouraged me to keep studying to achieve my goals. And to my siblings, Juan, Stella, Luis and Eduardo who are also my role models and stayed by me through good and bad times.

To my dear friend Demetrio who has always given me his generous and priceless help all along my academic career; and specially to complete the writing of this dissertation in the role of editor.

I would like to offer my special thanks to my PhD adviser, Gabriel, who encouraged me to finish this thesis against all odds.

Also, I wish to acknowledge the help provided by Laura, Franco Paula, Javier, Luciana and Carlos.

And finally, to my long standing friends, Walter, Federico, Demetrio, Diego and Damián, for giving meaning to one of the most important words for me, friendship.

Contents

CONTENTS

List of Figures

LIST OF FIGURES

List of Tables

Chapter 1

Introduction

Syntactic structure has been widely studied with the aim of developing applications capable of understanding natural language. However, researchers have always dealt with the limitation that the syntactic structure is not directly found in observable instances of language. Within this context, syntactic analysis has come to play a central role in developing applications that can automatically obtain the syntactic structure of the language we hear or read. This is why syntactic analysis has acquired a leading role in the vast range of studies involving automatic natural language processing.

This thesis attempts to contribute to the study of natural language processing by developing different methods of optimization for the automatic analysis of natural language syntax. This contribution can be observed in the three research works presented in this thesis. Two of them improve the performance of existing automatic syntactic analyzers (parsers), while the third one develops a brand new multi-language parser that yields state-of-the-arts results.

1.1 Brief history of Natural Language Processing and Syntactic Approach

We present a succinct description of the tools used in the Natural Language Processing (NLP) area, their history and motivation. In particular, the formalism of Bilexical Grammars (BG) for dependency parsing is introduced. A more in-depth coverage about BG will follow in Chapter 2.

In human history, the importance of information management and acquisition has always been crucial. This is due to the fact that timely information is essential for any effective decision-making process. In the last fifty years, the media have acquired an overwhelming massiveness. This vast expansion was further accentuated in the 90s, when the Internet was first made available to mass audiences. Since then, the amount of information available as written texts has become overwhelming.

Although access to information is highly connected to the notion of power, the mere access to it becomes less relevant when the information cannot be fully processed or understood. The fact is that the availability of information has come to such extent that human beings cannot longer handle these volumes of information by themselves.

This phenomenon has brought to mind the possibility of using computers as means to overcome human limitations by developing computational models capable of processing large amounts of information. Nonetheless, computers themselves bump against another limit: the inability to understand human language.

In recent years, Computer Sciences have dedicated lots of effort to research different ways of making it possible for computers to interpret human language. This approach towards language understanding is widely known as Natural Language Processing (NLP). Since the 80s, Natural Language Processing has taken a leading role within the scientific community.

To achieve its goal, an NLP system needs to create computer models that can handle the structure of language. Usually, these models do so at three different levels: morphological, semantic and syntactic. The morphological level deals with information related to the structure and content of morphemes and other units of meaning such as words, affixes and parts of speech (POS). The semantic level, attempts to understand the meaning and relation of signifiers, such as words, phrases and signs, in language. Finally, the syntactic level models the principles and rules by which words are combined to build a sentence.

The rest of this section summarises the development of NLP; the theoretical formalisms coming from linguistics as well as technological devices applied to natural language processing with focus in syntax.

The first electronic computers were built in the 40s, during World War II. In those years, researchers were focused on building a system capable of translating messages from and into different languages. Such computer programs were the first to process natural language. This decade models only took into account the order of words within sentences, and translated the text literally, looking up the words in a dictionary. The results obtained at that time were very poor. This low level achievements were believed to be, principally, due to the simplicity of the language model used.

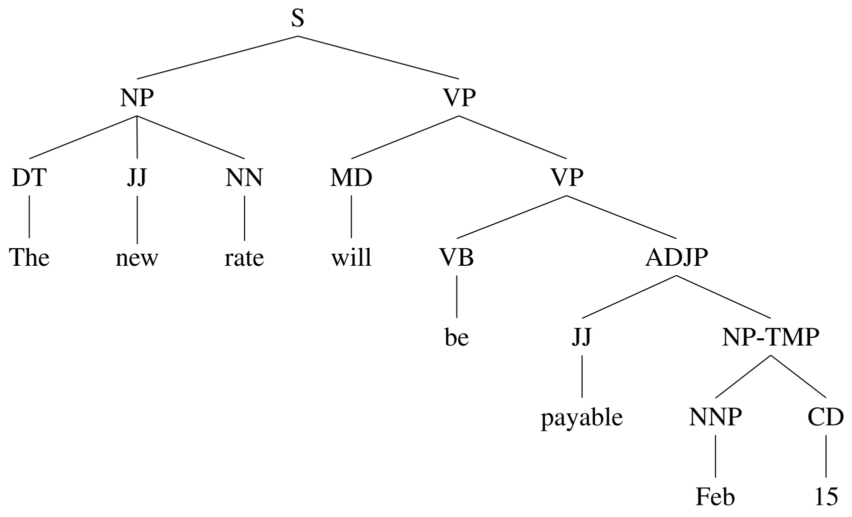


Figure 1.1: Phrase structure tree extracted from the Penn Treebank (MS93). This is a widely used set of trees annotated by linguists.

As a consequence of the poor results obtained in the first systems, NLP researchers started focusing their studies on trying to model natural language in a more formal way. Besides the effort made in these respects in the following years, significant progress in this field had to wait until the 60s, when Chomsky published (Cho65). In this work, Chomsky developed a grammatical theory to formalize natural language based on the notion of generation. Chomsky proposed a hierarchy of grammars capable of both generate the strings of languages starting from an initial symbol and mimic the syntactical structure of sentences through that generating process. Among Chomsky's grammars, the so called Context Free Grammar (CFG) were widely used for modeling syntactical phenomena of natural languages[1]. A CFG is a formal grammar where each production rule is of the form $N \rightarrow w$ where N is a single non-terminal symbol and w is a string counting one or more terminal or non-terminal symbols. In the NLP area, a syntactic tree is usually thought as the derivation tree of a certain CFG. More specifically, a derivation tree is thought as the result of the application of CFG rules to generate the sentence. In detail, each label of level i in the tree, is the left side of a CFG rule and the labels of the level $i + 1$ are the corresponding right side of the rule. For example, the tree in Figure 1.1 is gen-

[1]Of course, Chomsky never expected CFG could fully capture the syntax of natural languages. Moreover, in his book he explains the limitations of CFGs and the features of natural languages that they are unable to express.

erated by a CFG which starts with the application of a rule $S \rightarrow NP\ VP$, followed by the application of the next two rules$NP \rightarrow DT\ JJ\ NN$ and $VP \rightarrow MD\ VP$. The process continues applying CFG rules until the leaves, which contain a word of the natural language, are produced.

This formalization supposes that the syntactic structure of a sentence is organized in phrases hierarchically regulated. In Figure 1.1, there is an example of an English sentence organized hierarchically in phrases. The syntactic structure is represented using a tree. The root node spans the entire sentence. The rest of the nodes represent the sub-phrases of the sentence. For example, we can observe that the entire sentence (the node S) is divided in two sub-phrases: a noun phrase (node NP) and a verb phrase (VP) . The“leaves” of the trees are the words of the sentence. As having the full vocabulary of the language might be computationally costly, the actual words are often pruned. Thus, Parts Of Speech (POS) tags that represent the syntactical category of each word become the leaves of the tree[1]. In the tree of Figure 1.1, the node located before a word is the part of speech tag. For example, the word “will” have the MD POS tag which expresses that it is a modal verb. Usually, these kinds of syntactic representations are called phrase structures trees or constituent trees.

A few years later, as a reaction to Chomsky’s theory and following the concepts of Schank (Sch72), Mel’čuk developed an alternative theory for the syntactic analysis of natural language: the dependency grammar (Mel79). In this work, the author claims that the constituent formalism describes how the elements of language are combined to create larger elements, but it does not explain how the elements relate to each other. Alternatively to the phrase structure tree, the dependency formalism uses dependency trees.

An example of a dependency tree is introduced in Figure 1.2. In this directed tree, we see that each node is a word from a sentence and the nodes are ordered according to their position in the sentence. The arcs in that tree reflex the syntactic relationships between each word in the sentence. For example, in SVO[2] languages, the root of a dependency tree is the main verb and its left dependent is the main noun of the subject. If in a dependency tree there exists an arc from the word b pointing to the word a, we say that b is a dependent of a.

As linguistic theory progressed toward formal approaches, advances in electronics made computers practical and algorithms for NLP were devised and tried.

[1]A grammar using words as terminal symbols is said to be “lexicalized”.

[2]Languages are labeled as SVO, SOV . . . according to the prescribed order of main constituents of the sentences (Subject, Verb, Object).

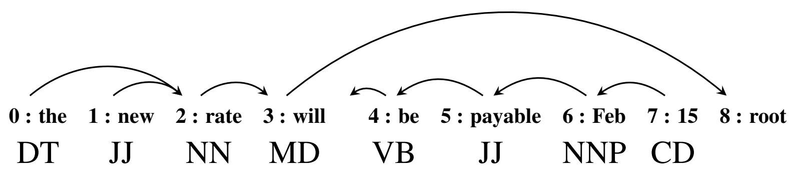


Figure 1.2: An example of a dependency tree.

Following Chomsky's work, Younger (You67) presented the the CockeYoungerKasami (CYK) , an efficient bottom up parsing algorithm for Context Free Grammars. As a consequence, in the 70s and early 80s the syntactic parsers (Ear70a, Mar80, Ram85) were based on CYK algorithm.

At those times, the approaches to syntactic parsing were symbolic. This means that they were based in deep analysis of linguistic phenomena. In general, the phenomena are encoded in a formal grammar [1] whose rules were defined by humans (for example, the rules created by Chomsky in (Cho65)).

In the nineties, a huge amount of text was available in digital format and the computational power grew even stronger; so a new, empirical, approach was tried. Instead of deriving the rules of natural language grammars from the work of linguists (to a large extent aprioristic), researchers looked in the texts actually written and took them as big sets of examples from which grammatical rules could be simply read. As an additional benefit, patterns of the frequencies with which rules were used could also be extracted from these texts. This led to the use of statistical models to approximate linguistic phenomena. Statistical methods look at the frequencies of different phenomena in actual utterances in order to infer probabilistic models of languages. It should be noted that statistical models often make much weaker assumptions about the underlying structure of the language and incorporate definite estimations of the probabilities of occurrence of certain phenomena. From the 80s until the mid 90s, the statistical approach was widely used in syntactic parsing, even the one that does not use a grammar to build a model (MM90). However, the best parsing models, including state-of-the-art ones (Col97, KM03a, Cha00), can be considered to a certain extent as hybrid. Although they use a grammar, most of their rules are learned from a set of examples made of annotated sentences. The grammars used by these parsers are called probabilistic, because their rules have a probability associated. In general, these probabilities are proportional to the fre-

[1]It can be a Context Free Grammar (Cho65)

quencies of data occurring in the examples. One kind of grammars widely used during those years was the Probabilistic Context Free Grammar (PCFG).

The examples used by the statistical methods are commonly called treebanks. These banks of trees are sentences which are usually annotated with syntactic information by a linguist. These treebanks are used to evaluate the syntactic parsers and in some cases to learn the model to parse. For English, one of the most used treebanks for phrase structure parsing is the Penn Treebank (PTB) (MS93).

Over the last years, the dependency formalism has become crucial for NLP syntactic analysis. Dependency structures are used not only to build dependency parsers, but they are also proved to be particularly useful in constituent parsers as Wang argues in (WZ10): the complexity of parsing algorithms for dependencies is better than that of constituent parsing[1]. Then efficiency of constituent parsing can be improved using dependency trees as an intermediate result; lexicalized PCFGs which use information about the head of constituents (Col96, Col97) perform better than those that lack this information. It is believed full information about the dependency structure can improve parsers' performance further.

For using dependency information in constituent parsing, the newest algorithms usually need to transform the phrase structure trees into dependency ones. For example, in (DFM04) there is a complete study on how to transform constituent trees into dependency ones. As the authors explain, for this transformation it is necessary to have some way to choose the most important word for each sub-phrase in the constituent tree. The most significant word in a constituent is usually called the head of this constituent. For example, the head of the constituent containing only the subject of the sentence is the main noun. For choosing the head in each constituent, a table that contains rules for each type of constituent (S, NP, VP, etc) is usually used. For example in Figure 1.3 the drawing at the bottom shows the dependency tree obtained transforming the tree at the top.

Another step in the development of NLP was the adoption of methods from machine learning. In this area, (some) learning algorithms are classified as supervised or unsupervised. For the parsing problem, supervised methods try to infer a grammar from the structure annotations of a training corpus of sentences. Unsupervised methods seek the underlying grammar that generates a set of utterances without annotations. This problem is obviously much harder, but given the high cost of annotating a corpus manually, a lot of research is currently focusing in this kind of algorithms (CGS08, CS09, HJM09, SAJ10a, GGG$^+$10, BC10, WZ10).

[1]Linear time vs. cubic time for CYK algorithm.

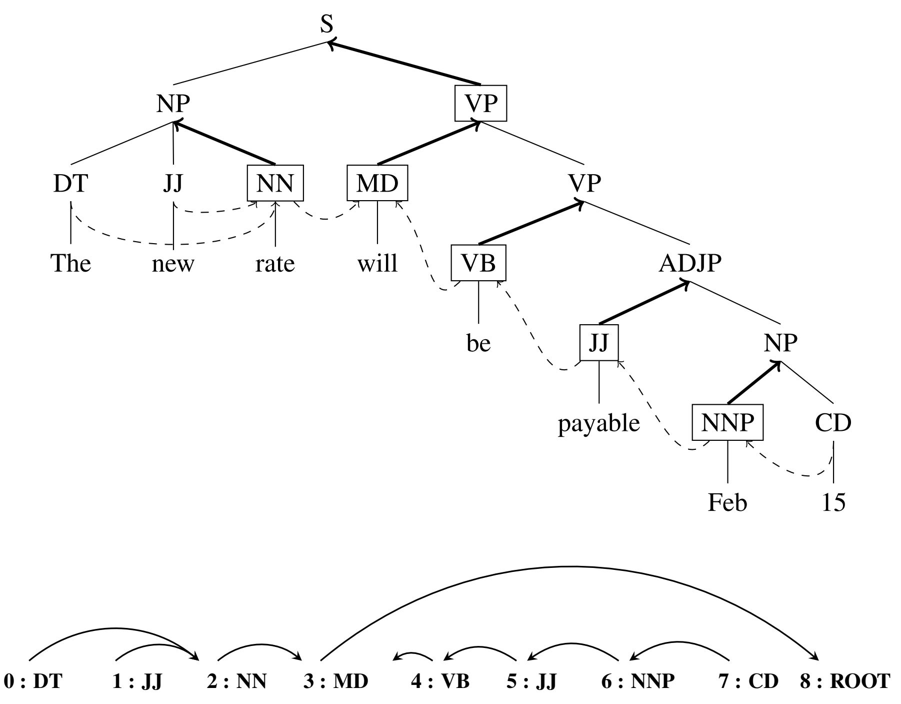


Figure 1.3: In the top of the figure, the phrase structure tree have marked head word at each level, the dashed lines show the dependency relation between the head words. In the bottom part of the figure, we found the dependency tree associated to the tree in the top according with these choose of heads.

1.2 Bilexical Grammars for Dependency Structures

Dependency structures are used to perform many different tasks in natural language processing. In particular, dependency structures are very useful in phrase structure parsing and dependency parsing, for both supervised and unsupervised methods. In this kind of theories, structure is modeled through a dependence relation between words. This relation carries information about headness and can overcome difficulties arising from contiguity constraints in phrase structure grammars. Most constituent supervised parsers (Col97, KM03a, Cha00) use dependency structures in their models to keep information of heads in sentences, and with this information they achieve better performance. Dependency models were also used in unsupervised parsers (Kle05, HJM09, PBE11, Seg07, Bod06).

Given a dependency treebank, we can define the dependency language L of this treebank. A word $W = l_1 \dots l_k \ * N \ r_1 \dots r_l$ belongs to L if N is a node in certain dependency tree with the following properties:

- l_i is dependent of N, $1 \leq i \leq k$
- r_j is dependent of N, $1 \leq j \leq l$
- for each occurrence of a word, all of its dependents are listed
- the order of the occurrences of dependents is preserved. That is if l and l' are both left dependents of w and l occurs before l', the subscript of l should be lower than the subscript of l' in the word corresponding to w.

Notice that we are linearizing dependency trees by listing the left and right immediate dependents for each word. The word under consideration is marked with the special symbol $*$.

For example, for the tree in Figure 1.2, the dependency words associated with the first four POS of the sentences are:

$$\{* \, DT, \ * \, JJ, \ DT \ JJ \ * \ NN, \ NN \ * \ MD \ VB\}$$

In this work we use Bilexical Grammars (Eis97) for modeling dependency languages. A Bilexical Grammar (BG) is equivalent to a Context Free Grammar (Eis96), but with a distinctive property: it is built from a set of regular languages. The idea behind this is to divide the entire dependency language into smaller sets to model them later. In general learning smaller languages is simpler, even more if they are regular. More precisely, we model dependencies for each word in the

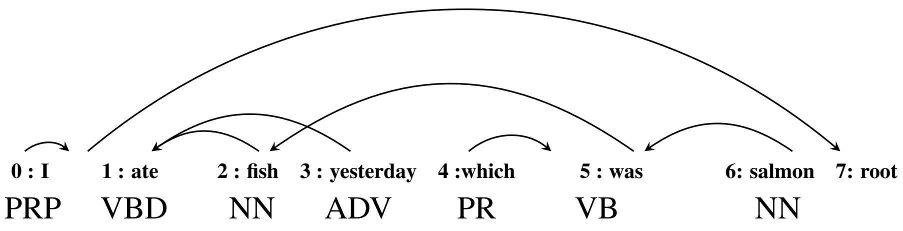


Figure 1.4: An example of a non-projective dependency tree.

lexicon splitting the right and the left dependencies. Once we have the right and left regular grammars for each word, we put all together with a Bilexical Grammar.

Given that regular languages are equivalent to the languages recognized by finite automata, the regular languages of left and right dependents of each POS in a Bilexical Grammar can be modeled in such way. As a consequence, we can say that a BG can be learned by inducing its automata. The strong hypothesis behind using a Bilexical Grammar to model dependencies structures is to assume that for each word, its left and right dependency languages can be accurately approximated by regular languages. This hypothesis is validated with the results obtained in three different experiments where dependencies are modeled with BGs. Of course, we can also make the standard addition of probabilities to the rules of the grammars (or, equivalently, add the weights to the transitions of the automata) giving rise to Probabilistic Bilexical Grammars (PBGs).

A characteristic of a dependency tree that was not mentioned before is that a dependency tree can be projective or non-projective. A projective dependency tree is one in which there are not crossing edges.

For example, the tree in Figure 1.1 is projective. In contrast a non-projective tree is the one shown in Figure 1.4. Here the relative clause “which was a salmon” and the object it modifies (“fish”) are separated by an adverb (“yesterday”). There is no way to draw the dependency tree for this sentence in the plane without crossing edges.

In this thesis, we consider only projective dependencies. This is because the bilexical grammar cannot emulate crossing lines by construction. A generalization of BG which enables the representation of projective trees is found in (MPRH05). However, as this generalization of BGs is not represented by automata, it is out of the scope of our thesis.

1.3 What This Thesis Is All About

This thesis, widely speaking, introduces different ways to capture the regularities of syntactic patterns of languages and apply them by computational means. We explore the behavior of dependency structures through the formalism of Probabilistic Bilexical Grammars and study their automata and their languages. This exploration is also done with strong use of computationally intensive procedures.

We address three main issues: how to obtain a "good" set of syntactical categories; how the manners of finding the leading words (heads) of constituents influence the performance of parsers and; how to infer grammatical structure of sentences in absence of explicit annotations. The use of PBGs across the different aspects of this study and the good results achieved gave us some insight about why bilexical grammars work.

For the problem of optimizing the alphabet of syntactical categories, we should point that these alphabets were chosen in somewhat aprioristic ways informed by linguistic tradition. It may be the case that better performance can be achieved if the choice of the set of syntactical categories is determined by the distributional properties of the (classes of) words. We add new symbols to the dependency language by splitting the POS tags. This idea is based on the fact that the syntactic category represents many different words and these words may have different, but by no means arbitrary, distributions. For this reason, the hypothesis is that we can learn dependency languages more easily by looking at the left and the right regular languages associated to each POS. For example, "I" and "them" are both pronouns, but the former appears more often before the verb than the latter.

This algorithm is principally used for verbs. The reason to choose verbs is that, in general, verbs are the most important words in the sentence, and therefore the dependency relationships are more complicated. Besides, verbs differ in the kind of arguments they usually take and show noticeable differences in usage.

We define an optimization method to find the best way of clustering the selected POS. We start splitting a POS according to a certain feature that encodes syntactic information. If we choose as a possible feature the POS of the left dependent, the initial cluster will have as many elements as different POS appear as left dependents in the training set. In each step we join those elements in the cluster which have "similar" languages. We evaluate the grammar obtained in each step according to the mistakes incurred by the parser in a test set and the simplicity of the grammar[1]. Our optimization obtains as a result the "best" partition according to a certain

[1]The rationale behind this lies in that we try to capture general patterns of the syntax of a

feature selected. In order to evaluate the resulting tag set, we add this information into two different state-of-the-art phrase structure parsers.

For example, we can propose as a feature for a verb to be followed by infinitives and the algorithm will cluster verbs like *try* and *quit* accordingly as if they shared that behavior in the training corpus.

The second of the issues has to do with syntactic analysis. One of the most important tasks in the NLP area is the development of a syntactic parser. A syntactic parser is a software system that when fed with a sentence of natural language, it returns a structure which contains relevant syntactic information. The most recent constituent parsers (cDMMM06, KM03b, Bik02, Col97, Cha00) use dependency information in the rules of their grammars. The information codified in the rule is the most important word in each sub-phrase; in other words they use the head of each sub-phrase. In this work, we propose an optimization method to study the "best" way to relate the words in a sentence and hence obtain the heads of constituents. The idea is to implement it by changing the rules used by the constituent parser to choose the heads in each level of a phrase structure tree.

The first naive option would be to implement a brute force algorithm which traverses the space of all possibles head rules, and evaluates them based on the performance obtained by the constituent parser selected. However, this solution is impracticable, due to the huge amount of different ways of choosing heads. In fact, the longest rule has length 18, so the number of possible alternatives amounts to $18! * 2^{18}$. Our idea is to use genetic algorithms to select promising variations of head finding rules according to their performance and the simplicity of the underlying model.

In each step of our algorithm, we look for better versions of the rules one at a time. With these candidates for improvement, a new rule set is used to transform a training set of phrase structure trees into dependency ones. We infer a bilexical grammar from this new set of dependency trees. The BG obtained in each step is then evaluated in a way similar to our first approach. Note that, by changing the head rules, we are changing the languages of the automata. As a result of this algorithm we obtain a new set of head-rules. Our hypothesis is that with this optimization process we can find the most convenient set of head rules for constituent parsers. In addition, this is a good way to evaluate the new set of head rules.

Our third work, namely learning grammars starting from sets of plain sentences falls in the area of unsupervised dependency parsing. This is, we hope to find regularities in the languages by applying statistical methods to large quantities of data.

language. These patterns should not be too complex.

An example of the kind of linguistic patterns we should be able to infer from a corpus, consider that the dependant of a preposition is usually a noun occurring to the right of that preposition. A rule like that can explain -that is, parse- a number of different sentences which other candidate rules can not. Then, a sound method should prefer such a rule over competing alternatives and "discover" that grammatical rule.

The strength of unsupervised methods is that they don't require high quality in their input but are capable of getting useful information from sufficient quantities of raw sentences. Moreover, they provide good insights of theoretical interest on linguistics and machine learning. The downside of this is that they need rather heavy computational resources and available data.

As in the two previous works, our algorithm is based on probabilistic bilexical grammars. Our focus of this algorithm is on learning the best grammar by using constraints in the complexity of their regular grammars. This idea is implemented by fixing the structure of the automata whose languages correspond to the ones of the grammars.

Given that the task is unsupervised, the input is a set of non-annotated sentences. The grammar is learned from an initial set of trees, those trees are built based on constraints aimed to start the learning process from simple models. In each iterative step of the algorithm, we parse the set of sentences with the PBG and refine the grammar by contrasting the parsed trees of the input sentences. As before, we measure the quality of the grammar by means of a combination of performance and simplicity.

The results of our three works are close to the state-of-the-art of the syntactic parsing. We use dependency structures to improve the performance of state-of-the-art supervised parsers and we developed a new unsupervised parser also based in dependencies.

In the first approach, we obtain a split of the POS tags by clustering words whose automata are "similar", and we treat each cluster as a new POS tag. We try out the resulting tag set in two different state-of-the-art phrase structure parsers and we show that the induced part of speech tags significantly improves the accuracy of the parsers.

In the second approach, we try to optimize the set of head finders rules used by phrase structure parsers to determine all heads in a sentence. We do so by changing the rules and evaluating them according to the languages of the automata inferred in the bilexical grammar induced with those rules. In this optimization process we look for the best language to be learned.

In our third work we build a novel unsupervised dependency parser whose flexibility of changing the structure of automata results in good performance in eight different languages. We combine tests in different languages and different automata structure and we obtain state-of-the-art results for English, Swedish, German, Turkish, Bulgarian, Spanish, Portuguese and Danish. We also find out that our parser engine can obtain competitive results even with a small amount of input sentences. We discover an unsupervised way to decide which automata structure is suitable for a given amount of sentences.

The results obtained by our three works show that Bilexical Grammar is a highly useful formalism to represent natural language dependencies. Results indicate that the use of information about heads leads to significant improvements in parser performance while variations of head finding algorithms are not so important. We conclude that the power of Bilexical Grammar lies mostly in its flexibility. The bilexical grammar allows us to access to the dependency language of each word and decide which learning mechanism is better to use in a particular word and in different training sets sizes. This flexibility allows us to decide, according to the task, which aspect of the dependency structures must be taken into account to achieve the better solution for a task.

1.4 The structure of this thesis

This thesis is structured as follows: in the next chapter we include some background necessary to understand in detail the rest of this work. In chapter 3, we present our approach to optimize the POS set for phrase structure parsing and how it improves the results of Bikel's implementation of Collins' parser (Bik02).

In chapter 4, we present in detail the head finder rules optimization for constituent parsing and we show how the performance of dependency parsers depends strongly on the existence of a head finding phase, but the actual rules used in that process have a much less significant impact.

Chapter 5 presents our unsupervised dependency parser. Our algorithms are full-fledged and easily reproducible. We experiment in eight languages (English, Swedish, German, Turkish, Bulgarian, Spanish, Portuguese and Danish) that inform intuitions in training-size dependent parametrization.

Each of the three later chapters ends with a discussion about the placement of each work in the context of current NLP research. Chapter 6 concludes our work.

1. INTRODUCTION

Chapter 2

Theoretical Framework

This chapter presents the theoretical background that is necessary to understand the different notions introduced in this thesis. First, we start by defining all the concepts related to syntactic analysis, phrase structure analysis and dependency analysis. Then, we explain the optimization algorithms used in this thesis.

2.1 Syntactic Analysis

The "Syntax" studies the regularities and constraints of word order and phrase structure (Niv05). In a sentence of natural language, at first sight, we can only observe a one-dimensional relation between words, i.e., the sequence as an ordered list of strings. However, the central idea of a syntactic theory is to uncover the hierarchical relationship between words. Given that natural language is ambiguous, this hierarchical structure can help to interpret the meaning of a sentence (MS02). For example, the sentence "She saw a man with a telescope", could have two hierarchical structures: one in which the telescope is related to "she" and another syntactic analysis where "the telescope" is related to "man".

Syntactic parsers are considered of great importance in the area of Natural Language Processing. These algorithmic systems can analyze natural language and identify the different functions words may have within a sentence. Moreover, they are designed to break down a phrase into a tree structure and add the different grammatical categories. The relevance of the syntactic analysis lies in the fact that their results are applicable to multiple areas of study such as automatic language generation, automatic translation, voice recognition, and many others.

2.1.1 Phrase Structure Parsing

Natural language has an underlying structure usually referred to as Syntax. The fundamental idea of syntax is that words group together and form constituents, which are groups of words or phrases that behave as a single unit. These constituents can combine together to form larger constituents and, eventually, sentences. The function of a Phrase Structure Parser is to discover the tree structure of a given input sentence by detecting the sub-phrases that form such utterance and defining how they are hierarchically related. One of the characteristics of constituents is that they can occur in different positions. For example:

- She left the books on the desk.
- She left on the desk the books.
- On the desk, she left the books.

Another characteristic of constituents is that they can be replaced by a shorter sequence of words while the sentence remains grammatically correct. To clarify this idea, consider the following sentences:

- John and Terry walked through the park in a hurry.
- They walked through the park in a hurry.

The fact that we can substitute “John and Terry” by “They” shows that the former group of words is a constituent. On the contrary, “ John and Terry walked” cannot be replaced in the same way, as we can see in the following sentence:

- They through the park in a hurry.

Within the context a Phrase Structure Tree, such as the one in Figure 1.1, each node of the tree (including the words) is considered a constituent. In a tree view, constituents at lower levels join together to form larger constituents. The higher levels of constituents (above POS tag nodes) are the grammatical categories. The most important grammatical categories are (MS02):

- **Noun phrases.** The noun usually embedded in a noun phrase (NP) is considered the head of the noun phrase; the main constituent that defines its syntactic function. This syntactic unit provides information about the head noun. Noun phrases usually function as the arguments of verbs. i.e., they represent the participant in the action, activity or state described by the verb. Noun phrases normally consist of the following elements: a determiner (optional), adjectives phrases (optional), a noun head, and they may contain some post-modifiers, such as prepositional phrases or relative clauses. The constituents of a noun phrase usually appear in the order mentioned. Here is a large noun phrase that includes many of the possibilities mentioned:

  - *The old country house on the hill* is being remodeled.

- **Prepositional phrases.** Prepositional phrases (PPs) are led by a preposition and contain a noun phrase which is the object of the preposition. They can appear within all the other major phrase types. They are usually found in noun and verb phrases where they normally express circumstantial attributes (space, time, manner, etc.). For example:

  - The girl *in the blue dress* is dancing *with me*.

- **Verb phrases.** The verb is the head of the verb phrase (VP). In general, the verb phrase rules all elements of the sentence that depend syntactically on the verb. Some examples of verb phrases are:

  - He *explained his master plan thoroughly*.
  - He *was fighting for his freedom*.
  - The woman *eats fruit*.

- **Adjective phrases.** Adjective phrases (APs) are less common. They consist of an adjective that functions as the head of the phrase and may contain other elements such as modifiers, determiners, and qualifiers. Some examples of such phrases are:

  - He was *quite sure* he saw her last night.
  - They were *much more grateful* than we expected.

2.1.1.1 Phrase Structure Grammars

The formalization of the constituent parsing was introduced by Chomsky in (Cho53, Cho57) by using Phrase Structure Grammars (PSG). A phrase structure grammar is defined as a four-uple (S, T, N, R) such that:

- T, the set of terminal symbols, the words of the language being defined.

- N, the non-terminal symbols: these are the union of two subsets $GC \cup POS$ where GC are the grammatical categories and POS are the syntactical categories.
- R, a set of productions of the form $V \rightarrow W$, where V is a sequence in $(T \cup N)^+$ and $W \in (T \cup N)^*$ is a sequence of zero or more symbols from either alphabet. If V is just a member of POS, the right hand side W must be a member of T.
- S, a start symbol, a member from N, that is a grammatical category.

A Phrase Structure Grammar is called Context Free when each of the productions in R is of the form $V \rightarrow W$ where V is a single symbol in N. For example, Table 2.1 shows a very simple CFG phrase structure grammar.

For CFGs we can define derivation trees which can give us a graphical representation of how a sentence can be obtained by applying the production rules to the starting symbol. Formally, a derivation tree for the sentence w in the grammar (S, T, N, R) is an ordered labeled tree such that:

- The root of the tree is labeled S.
- The labels of the leaves belong to $T \cup \{\epsilon\}$.
- The labels of non-leaves nodes belong to N, and if a node has label L and the labels of its children are $(L_1, ..., L_n)$, then $L \rightarrow L_1...L_n$ must be in R.
- w is the concatenation of the labels of the leaves.

We can now define the *string language* generated by a CFG G as the set $L(G) = \{w : a\ derivation\ tree\ for\ w\ in\ G\ exists\}$. Similarly, we define the *tree language* of G is the set of all possible derivation trees.

Figure 2.1 plots a constituent tree, where in each node, bellow the sintactic category, it is shown the PSG rules from Table 2.1 that were used to derive the sentence.

A CFG is considered *ambiguous* if a sentence in its string language has at least two different derivations trees. The CFGs used to model natural languages are usually *highly ambiguous*. Therefore, it is necessary to have a method to decide which of the many derivation trees of a given sentence captures best its true meaning.

For CFGs a well-known result is that for every CFG G, there is another CFG $G' = (S, T, N, R)$ such that $L(G) = L(G')$ and G' has the following properties:

S → NP VP	
NP → DT NN	DT → the
NP → NP PP	NN → man
NP → PRP	NN → telescope
VP → VP PP	IN → with
VP → VBD NP	PRP → she
PP → IN NP	VBD → saw

Table 2.1: A simple CFG grammar.

- If $\epsilon \in L(G')$ then $S \to \epsilon \in R$
- Each other production in R is of one of the following forms:
  - $A \to BC$, with $A \in N$ and $B, C \in N - \{S\}$
  - $A \to w$, with $w \in T$

A grammar having the properties mentioned is considered to be in Chomsky normal form (CNF). As there are algorithms to find the Chomsky normal form for CFGs (HU79), some algorithms that use CFG as input assume, without generality loss, that those grammars are in CNF. A relevant example of such algorithm is the CYK algorithm for parsing which we will describe later on.

We summarize how CFG can model natural languages saying that non-terminal symbols encode syntactical the categories of words as well as the ones of the constituents of the language to be modeled; the production rules capture the permissible ways of combining constituents. Finally, the derivation trees provide a description of the underlying syntactical structures of sentences.

2.1.1.2 Symbolic Parsing

Given a context free grammar G and a sentence, the problem of deciding whether the sentence belongs to the string language of G and, if so, yielding one or more of its derivation trees is called syntactic parsing. In the case that there exist a derivation tree for a given sentence in the language of G, de returned derivation tree are usually called *parse tree*. This problem is known to be decidable and two basic approaches to it are top-down parsing and bottom-up parsing.

- In the top-down approach, a parser tries to derive the given string from the start symbol by rewriting non-terminals, one by one, using productions. The

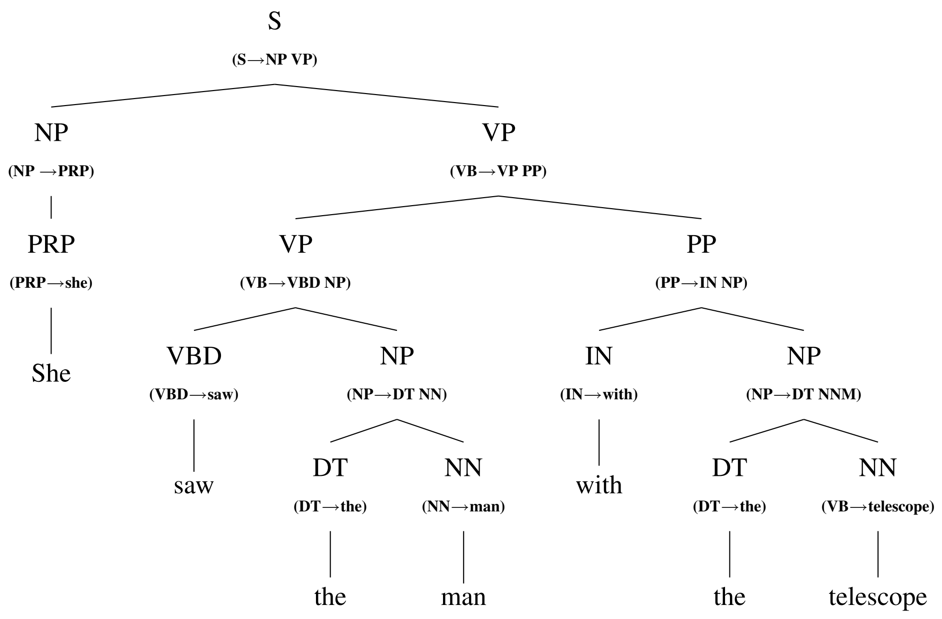


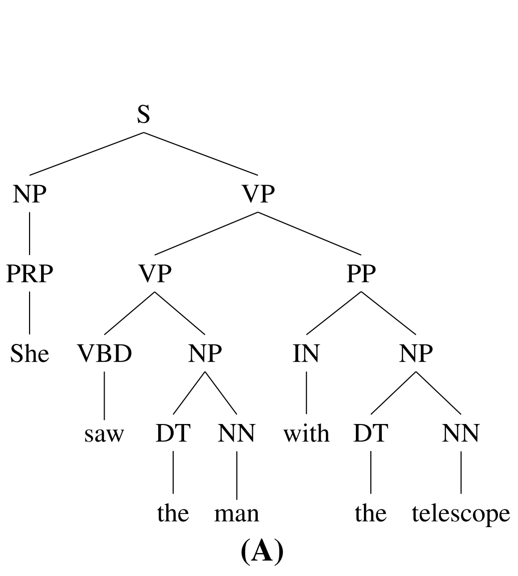
Figure 2.1: Phrase structure derived from grammar in table 2.1. Under each label of a node, the rules used to derive the tree for each of its levels are shown.

non-terminal on the left hand side of a production is replaced by its right hand side in the string being parsed.

The most popular and efficient top-down algorithm was implemented by Earley (Ear70b) using dynamic programming. The Earley parser has an $O(n^3)$ worst case time complexity, where n is the length of the parse sentence.

- In the bottom-up approach, a parser tries to transform the given string to the start symbol, step by step, using productions. The right hand side of a production found in the string being parsed is replaced by its left hand side. The first efficient parser for CFG was a bottom-up parser called CYK algorithm (You67). CYK stands for Cocke-Younger-Kasami, and such algorithm involves parsing sub-strings of length 1, then length 2, and so on until the entire string has been parsed. The reason why this algorithm is efficient is because the shorter sub-strings from previous iterations can be used when applying grammar rules to parse the longer sub-strings.

We will explain the CYK algorithm in detail because all of our parsers are based on this algorithm. The Algorithm 1 shows the CYK algorithm, it takes as input a grammar G and a sentence w. It considers every possible sub-sequence of the sequence of words and sets $B[l, r, a]$ to be true if the sub-sequence of words starting from i of length l can be generated from R_a. Once it has considered sub-sequences



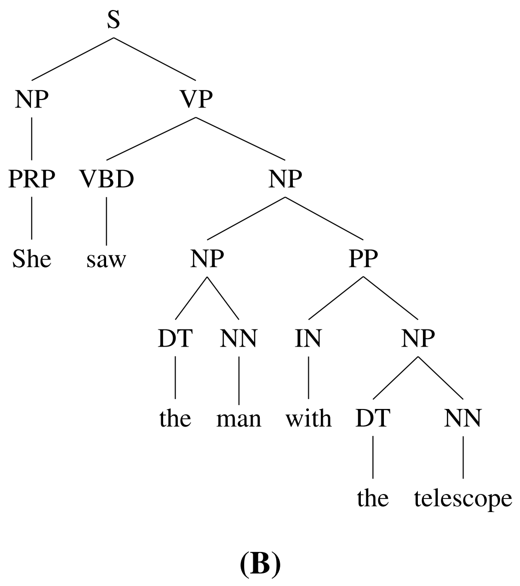


Figure 2.2: Two possible phrase structure trees for the sentence "She saw the man with a telescope".

of length 1, it moves to sub-sequences of length 2, and so on. For sub-sequences of length 2 and greater, it considers every possible partition of the sub-sequence into two parts, and checks to see if there is some production $X_a \rightarrow X_b X_c$ such that X_b matches the first part and X_c matches the second part. If so, it records X_a as matching the whole sub-sequence (from l to r). Once this process is completed, the sentence is recognized by the grammar if the sub-sequence containing the entire sentence is matched by the start symbol S.

It is easy to modify this algorithm to recover the parse tree. The tree nodes should be stored in the array B, instead of booleans. Then the array in $B[l, r, a]$ should keep the list of all non-terminals that generate the span tree from l to r.

2.1.1.3 Statistical Parsing

The CFG is a good formalism to find all possible phrase structure trees. However, this formalism does not solve the ambiguity; it produces all possible trees. Sentences can have many parse tree according to the CYK algorithm. At some point, to perform a task, we need to choose only one tree. For example, consider the trees in Figure 2.2, both of them are generated by the grammar in Table 2.1. In order to solve this type of ambiguity, NLP researchers started to use formal grammars which include probabilities: Probabilistic Context Free Grammars (PCFG) (BT73).

```
Data: the CFG $G = (S, T, N, R)$, $N \cup T = \{X_1, \ldots, X_k\}$, the
      input sentence $w = \{w_1, \ldots, w_n\}$
Result: true if there is a parse tree, and also $B$ contains the tree,
        false if there is not a parse tree
for $i = 1$ to $n$ do
    foreach $(X_j \rightarrow x) \in R$ do //iterate in rules
        if $x == w_i$ then //initialize the diagonal, for
        unary rules
            $B[i; i; j]$ =true
        end
    end
end
for $i = 2$ to $n$ do //set length of window for span
    for $l = 1$ to $n - i + 1$ do //starting the span
        /* the span window is $s = w_l w_{l+1} \ldots w_r$.     */
        $r = l + i - 1$;
        for $m = l + 1$ to $r$ do //Splitting the span
            /* span window $s = w_l w_{l+1} \ldots w_r$           */
            /* $x = w_l w_{l+1} \ldots w_{m-1}$, $y = w_m w_{m+1} \ldots w_r$, and
               $s = xy$                                            */
            foreach ($X_a \rightarrow X_b X_c \in R$) do
                /* Can we match $X_b$ to $x$ and $X_c$ to
                   $y$?                                            */
                if $B[l, m-1, b] \wedge B[m, r, c]$ then
                    $B[l, r, a]$ =true;
                end
            end
        end
    end
end
if $B[1, 1, n]$ then asda
    return true;
end
else
    return false;
end
```

Algorithm 1: The algorithm $CYK(G, w)$.

A PCFG is simply a CFG with probabilities added to their rules, indicating how likely different rewritings are. Formally, a PCFG is 5-uple (S, T, N, R, P)[1] such that:

- T, the set of terminal symbols, the words of natural language being defined.
- N, the non-terminal symbols: these are the union of two subsets $GC \cup POS$ where GC are the grammatical categories and POS are the syntactical categories.
- R, a set of productions of the form $w \rightarrow W$, where w is a non-terminal symbol and $W \in (T \cup N)^+$ is a sequence of one or more symbols from $T \cup N$.
- S, a start symbol, a member from N, that is a grammatical category.
- P, the probability, is a function with domain in R and image in the interval $[0, 1]$, such that,

$$\forall X \in N \qquad \sum_{y_1 \ldots y_n \,:\, X \rightarrow y_1 \ldots y_n \in R} P(X \rightarrow y_1 \ldots y_n) = 1$$

In the rest of this work, for a grammar $G = (S, T, N, R, P)$ the following notation will be used: $Rules(G) = R$, $Term(G) = T$ and $NTerm(G) = N$.

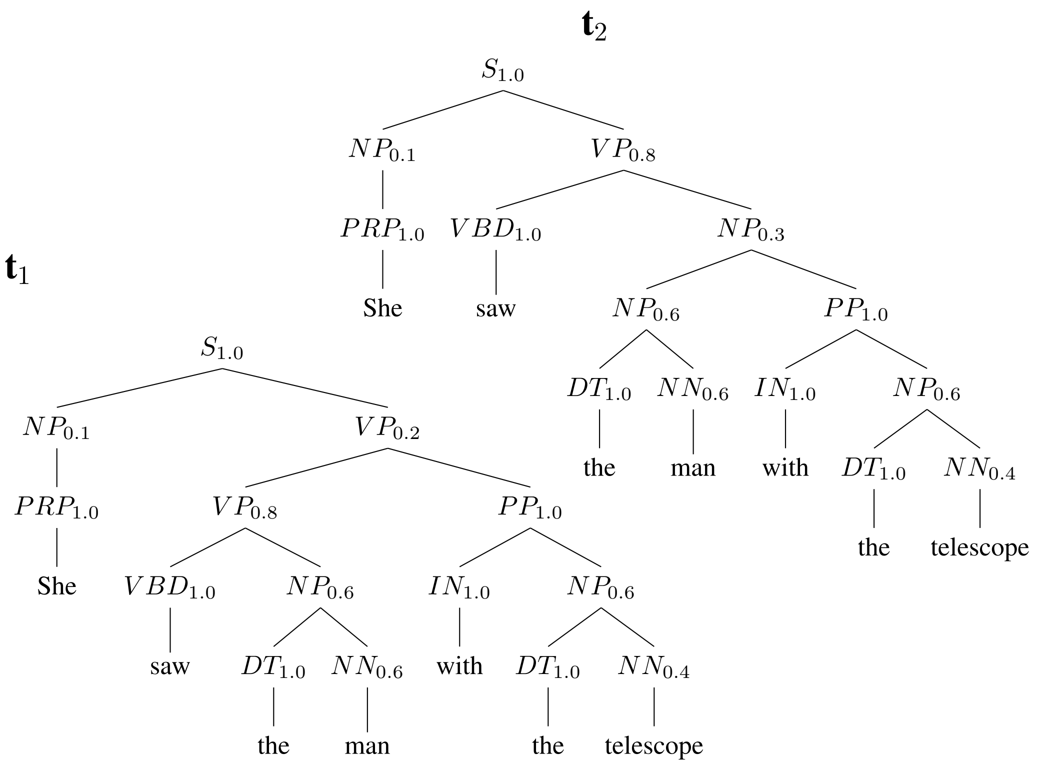
For example, Table 2.2 shows a PCFG grammar based on the rules of CFG used in previous examples (Table 2.1). Using this PCFG we can show how to solve the ambiguity in these sentences if the task requires to do so. The probability of a tree is defined as the product of the probabilities of rules used to derive the parse tree. As a consequence, using the rules in Table 2.2, Figure 2.3 shows how the PCFG can help to solve the ambiguity. Considering the example mentioned, "with the telescope" is more likely to be attached to "the man".

Gold in (Gol67) showed that CFGs cannot be learned without the use of negative evidence (the provision of ungrammatical examples). He used the concept of learning in the sense of identification in the limit - that is, whether one can identify a grammar if one is allowed to see as much data produced by the grammar as one wants. However, PCFGs are good for grammar induction, as they can be

[1]It is expected that the sum of probabilities of all trees produced by a PCFG G to be 1. But this is not always the case. However, as long as we only compare probabilities between trees this is not a relevant constraint. Finally, some works which study PCFGs with the consistency property can be found in (SB97, Chi99, NS06).

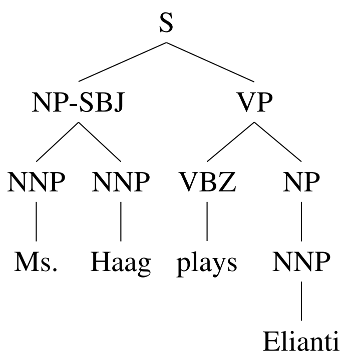
S → NP VP	1.0		
NP → DT NN	0.6	DT → the	1.0
NP → NP PP	0.3	NN → man	0.6
NP → PRP	0.1	NN → telescope	0.4
VP → VP PP	0.2	IN → with	1.0
VP → VBD NP	0.8	PRP → she	1.0
PP → IN NP	1.0	VBD → saw	1.0

Table 2.2: A simple PCFG grammar.



The probabilities of $\mathbf{t}_1$ and $\mathbf{t}_2$:
P($\mathbf{t}_1$)= $1.0{*}0.1{*}0.2{*}1.0{*}0.8{*}1.0{*}1.0{*}0.6{*}1.0{*}0.6{*}1.0{*}0.6{*}1.0{*}0.4 = 0.0013824$
P($\mathbf{t}_2$)= $1.0{*}0.1{*}0.8{*}0.1{*}1.0{*}0.3{*}0.6{*}1.0{*}1.0{*}0.61.0{*}0.6{*}1.0{*}0.4 = 0.00020736$

Figure 2.3: Solving the ambiguity with the PCFG in Table 2.2 for sentence “She saw the man with the telescope”.



```
(S
   (NP-SBJ (NNP Ms.) (NNP Haag) )
   (VP (VBZ plays)
     (NP (NNP Elianti)))
   (. .) )
```

Figure 2.4: Sentence number 2 from Section 2 of the Penn Tree Bank. The left figure shows the graph of the tree, and on the left, the PTB style annotation.

learned from positive data alone (Hor69). For syntactic parsing, the positive examples for learning a PCFG model for parsing are called treebanks. For phrase structure parsing, usually these treebanks are a set of sentences where the constituents are marked and labeled with their syntactical and grammatical categories. The most popular treebank for English is the Penn Treebank (PTB) (MS93). The PTB is a set of English sentences which are annotated by linguists. These sentences are organized in sections. The style of the annotation used is "LISP style"[1], bracketing the constituent. For example, on the right of Figure 2.4 there is an example of such sentences and, on the left, we draw the tree represented by the PTB annotation. In Table 2.4 we show the syntactic and grammatical categories used in the PTB annotation, while in Table 2.3 the POS tags used are listed.

Other well-known corpus used for constituent parsing are: for English (FK79), for German (BDH$^+$02) and for Spanish (CMBN$^+$03), among others.

2.1.1.4 Treebanks PCFGs:

When the PCFG is induced from a corpus such as the PTB, this type of learning a grammar is called **Supervised Learning**. This is because we want to learn a grammar, and we have positive examples of the rules of the grammar to be learned.

When we learn a grammar from a treebank, we take into account the rules observed in the trees of the training material, and the probabilities of the rules are estimated according to the Maximum likelihood estimation (MLE). That is, MLE estimates from relative frequencies from a given treebank T:

[1]The tag of the considered constituent is in the place of the function name in the LISP style writing. Recall that the function $f(x, y)$ is written as `(f x y)` in LISP.

Num.	Tag	Description
1	CC	Coordinating conjunction
2	CD	Cardinal number
3	DT	Determiner
4	EX	Existential there
5	FW	Foreign word
6	IN	Preposition or subordinating conjunction
7	JJ	Adjective
8	JJR	Adjective, comparative
9	JJS	Adjective, superlative
10	LS	List item marker
11	MD	Modal
12	NN	Noun, singular or mass
13	NNS	Noun, plural
14	NNP	Proper noun, singular
15	NNPS	Proper noun, plural
16	PDT	Predeterminer
17	POS	Possessive ending
18	PRP	Personal pronoun
19	PRP$	Possessive pronoun
20	RB	Adverb
21	RBR	Adverb, comparative
22	RBS	Adverb, superlative
23	RP	Particle
24	SYM	Symbol
25	TO	to
26	UH	Interjection
27	VB	Verb, base form
28	VBD	Verb, past tense
29	VBG	Verb, gerund or present participle
30	VBN	Verb, past participle
31	VBP	Verb, non-3rd person singular present
32	VBZ	Verb, 3rd person singular present
33	WDT	Wh-determiner
34	WP	Wh-pronoun
35	WP$	Possessive wh-pronoun
36	WRB	Wh-adverbv

Table 2.3: POS tags from the PTB.

Num.	Phrase Tag	Description
1	S	Simple declarative clause, i.e. one that is not introduced by a subordinating conjunction and that does not exhibit subject-verb inversion.
2	SBAR	Clause introduced by a subordinating conjunction.
3	SBARQ	Direct question introduced by a wh-word or a wh-phrase.
4	SINV	Inverted declarative sentence, i.e. one in which the subject follows the tensed verb or modal.
5	SQ	Inverted yes/no question, or main clause of a wh-question, following the wh-phrase in SBARQ.
6	ADJP	Adjective Phrase.
7	ADVP	Adverb Phrase.
8	CONJP	Conjunction Phrase.
9	FRAG	Fragment.
10	INTJ	Interjection. Corresponds approximately to the part-of-speech tag UH.
11	LST	List marker. Includes surrounding punctuation.
12	NAC	Not a Constituent; used to show the scope of certain prenominal modifiers within an NP.
13	NP	Noun Phrase.
14	NX	Used within certain complex NPs to mark the head of the NP.
15	PP	Prepositional Phrase.
16	PRN	Parenthetical.
17	PRT	Particle. Category for words that should be tagged RP.
18	QP	Quantifier Phrase (i.e. complex measure/amount phrase); used within NP.
19	RRC	Reduced Relative Clause.
20	UCP	Unlike Coordinated Phrase.
21	VP	Verb Phrase.
22	WHADJP	Wh-adjective Phrase. Adjectival phrase containing a wh-adverb, as in how hot.
23	WHAVP	Wh-adverb Phrase. For example how or why.
24	WHNP	Wh-noun Phrase. e.g. Who, which book, whose daughter, none of which, or how many leopards.
25	WHPP	Wh-prepositional Phrase. Prepositional phrase containing a wh-noun phrase (such as of which or by whose authority).
26	X	Unknown, uncertain, or unbracketable. X is often used for bracketing typos.

Table 2.4: Constituent tags from the PTB.

$$\hat{P}(A \rightarrow s_1 \dots s_n) = \frac{Count(T, A \rightarrow s_1 \dots s_n)}{Count(T, A \rightarrow *)}$$

where $Count(T, r)$ is the number of times that the rule r appears in treebank T, and $Count(T, A \rightarrow *)$ means the number of rules with the non-terminal A in its left hand side. The problem with the MLE estimator is the data sparseness. That is, even with a large corpus there exist elements in the training corpus which are infrequent.

To solve this problem, usually "smoothing" techniques are applied (MS02). For example, those rules appearing less often than a certain threshold are treated as an only one rule. A similar strategy is used for words occurring just a few times. Smoothing can also be used when the parser receives a word which does not appear in the training data. In this case, such word will be treated as a non-frequent word.

To solve data sparseness, state-of-the-art parsers frequently implement the inside-outside algorithm (Bak79). This algorithm is also used to train PCFGs when there are unannotated treebanks, i.e. when only a set of sentences is available. This way of inducing grammar is called **Unsupervised Learning**, because the grammar is learned using as example elements that belong to the language of the PCFG, in this case the NLP sentences.

The standard procedure for learning both, supervised and unsupervised grammars, is to divide the available material in two:[1] the training set, which is used to learn the grammar, and another set for tests which is used to parse it and evaluate the results obtained.

2.1.1.5 PCFG parsing

The idea of parsing is to be able to take a sentence s and to work out parse trees according to some grammar G. In probabilistic parsing, we would like to place a ranking on possible parses showing how likely each one is (kbest-parsing), or maybe to just return the most likely parse of a sentence (best parsing). Given a PCFG $G = (S, T, N, R, P)$, it is important to explain the language that G defines. In this sense, the probability assigned to each sentence s that belongs to the language $L(G)$ is:

$$P(s) = \sum_{t:yield(t)=s} P(t|s)$$

[1]For supervised learning we have a set of annotated sentences, while for unsupervised learning we have just sentences.

where $yield(t)$ returns the sentence associated to the tree t. Additionally, using the definition of conditional probability, and also, using the independence assumption of PCFG, we can define the probability of a tree t for a given sentence s as follows:

$$P(t|s) = \frac{P(t,s)}{P(s)} = \frac{P(t)}{P(s)} = \frac{\prod_{r \in Rules(t,G)} P(r)}{P(s)}$$

where $Rules(t, G)$ returns the rules from grammar G used to derive the tree t.

Given this language model, the best parsing algorithm for a sentence s can be simply defined as:

$$\operatorname*{argmax}_{t:s=yield(t)} (P(t|s)) = \operatorname*{argmax}_{t:s=yield(t)} P(t) = \operatorname*{argmax}_{t:s=yield(t)} \prod_{r \in Rules(t,G)} P(r)$$

For solving this optimization problem, a variant of the original CYK is presented in Algorithm 2. Note that the only change this algorithm has with respect to the original CYK is found in lines 13 and 14. Once this algorithm computes the matrix B, to reconstruct the parse tree it is necessary to call the function $Reconstruct(l, m, r, B)$. In Algorithm 3[1], we can see the code to do so. Note that both algorithms can be implemented in only one, by storing the rule used in line 14 in the matrix B.

For implementing a K-Best parsing, the major change must be done in line 13 of the Algorithm 2. Instead of calling the max function, the algorithm must keep in $B[i, j, a]$ a list of the K best results of partial probability $P(X_a \rightarrow X_b X_c) * B[l, m-1, b] * B[m, r, c]$.

2.1.1.6 Lexicalized Grammars

The context-freeness property has desirable properties such as efficient parsing algorithms and intuitive modelization of constituency. Nevertheless, the restriction that production rules can be applied independently of the context introduces some unrealistic features in the models.

- PCFGs assume that the expansion of any non-terminal is independent of its parent. For example, $NP \rightarrow Pronoun$ is more likely for subjects (parent S) than objects (parent VP).
- The way in which a category is rewritten depends on things outside the subtree it dominates (plural verbs are more likely to occur with plural subjects).

[1]Both Algorithms 2 and 3 are variants of the ones published in `http://web.cs.dal.ca/~vlado/csci6509/notes/nlp21.pdf`.

```
Data: the PCFG $G = (N; T, R, S, P)$, $N \cup T = \{X_1, \dots, X_k\}$, the
      input sentence $w = \{w_1, \dots, w_n\}$
Result: the probability of the best parse tree for sentence $w$ in the
        grammar $G$.
1  for $i = 1$ to $n$ do
2      foreach $(X_j \rightarrow x) \in R$ do //iterate in rules
3          if $x == w_i$ then //initialize the diagonal, for
           unary rules
4              $B[i; i; j]$ =true
5          end
6      end
7  end
8  for $i = 2$ to $n$ do //set length of window for span
9      for $l = 1$ to $n - i + 1$ do
           /* the span window is $s = w_l w_{l+1} \dots w_r$ */
10         $r = l + i - 1$;
11         for $m = l + 1$ to $r$ do //Splitting the span
               /* the span window is $s = w_l w_{l+1} \dots w_r$ */
               /* $x = w_l w_{l+1} \dots w_{m-1}$, $y = w_m w_{m+1} \dots w_r$, and
                  $s = xy$ */
12             foreach $(X_a \rightarrow X_b X_c \in R)$ do
                   /* have better prob. $X_b$ to $x$ and
                      $X_c$ to $y$? */
13                 $B[l, r, a] = max(B[l, r, a], P(X_a \rightarrow$
                   $X_b X_c) * B[l, m-1, b] * B[m, r, c])$
14             end
15         end
16     end
17 end
18 return $B[1, r, 1]$
```

Algorithm 2: The algorithm $CYK_{PCFG}(G, w)$ is a generalization of the CYK algorithm for PCFG grammars.

```
Data:  $B$ the table from CYK - $i$ index of the first word cosidered in
       the sentence - $j$ length of the sub string considered - $a$ the
       index of the non-terminal which is the root of the tree to be
       reconstructed.
Result: the tree $t$ which $yield(t) = w_i \dots w_{j-1}$, and $root(t) = X_a$.
 1 if $j == 1$ then
 2     return the tree with root $X_a$ and child $w_l$;
 3 end
 4 for $l = 2$ to $j$ do
 5     foreach ($X_a \rightarrow X_b X_c \in R$) do
 6         if $(B[i, j, a] == max(B[i, j, a], P(X_a \rightarrow$
           $X_b X_c) * B[i, l-1, b] * B[l, j, c])$ then
 7             create a tree $t$ with root $X_a$;
 8             $t.left\_child = Reconstruct(i, l-1, b, B)$;
 9             $t.right\_child = Reconstruct(l, j, c, B)$;
10             return $t$
11         end
12     end
13 end
```

Algorithm 3: The algorithm $Reconstruct(l, m, r, B)$.

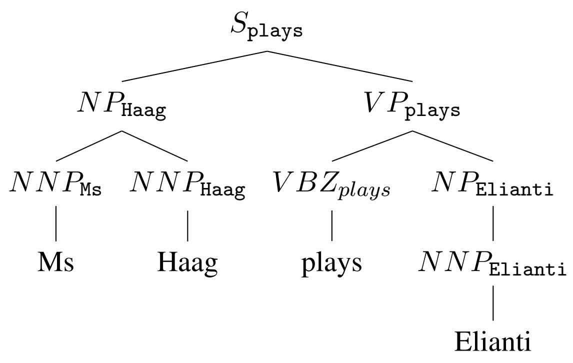


Figure 2.5: Lexicalized tree for a PTB tree.

- PCFGs ignore lexical content: green is more likely to be followed by leaf than by cat, or ideas, but PCFGs cannot capture this. How V is rewritten is independent of the choice of Subject and how the VP is expanded.

Some approaches (MM90, CCCC92, Mag95a, Col96, Col97) were proposed to deal with the problem of incorporating known features of natural languages to the models while keeping them simple enough to preserve their good properties related to efficiency and descriptiveness. A possible answer is with lexicalization, which means, adding to the non-terminal symbols, information about the words of sentences. One of the most popular lexicalized models are the Head Driven Grammars (PS94). This is the most straightforward and common way to lexicalize a PCFG by having each phrasal node marked with its head word. For example, Figure 2.5 shows how to lexicalize a tree from the Figure 2.4[1]. The process of learning a lexicalized PCFG is simple.

Lexicalization adds numerous new grammar rules, one for each lexical item added. A lexicalized PCFG can be learned from a corpus by adding the head information to their trees and then proceed as in standard PCFG learning algorithm. The CYK algorithm can be used for parsing the lexicalized PCFG in a straightforward way. State-of-the-art parsing algorithms (Col97, KM03a, Cha00), use head driven grammars or similar variants.

It is important to remark that it is possible to do a variant of Head Driven Grammars by using the head POS instead of the head word. These models are usually called Unlexicalized-Models.

[1]In Figure 2.5 we exclude from the grammatical categories the additional information usually included in the PTB trees after the symbol $-$. For example we use NP instead of $NP - SBJ$.

2.1.1.7 Evaluation

The accuracy in constituent parsing is usually measured using the *recall* and *precision* metrics. Given a set of parsed trees PT for a given set of sentences S, and the set of trees GT assumed as correct for the same set of sentences S, for each tree we have:

- *Recall* is the the number of correct non-terminal labeled constituents divided by the number of such constituents in the GT.
- *Precision* is the number of correctly identified constituents divided by the number of constituents produced in the parsed tree.

In other words, *precision* gives an idea of how much of what was produced is correct, while *recall* measures how much of what should have been produced was actually obtained. The idea of "correct" means that we compare the tree obtained by the model to be tested with the corresponding sentences in a gold-standard trees, assumed to be well-annotated. The *precision* and *recall* of the complete treebank PT is obtained as the mean of each of these measures calculated for each tree.

The comparison of the constituents considering the dominating non-terminal is called *labeled precision/recall*. If only the start and ending are considered it is called *unlabeled*. For example, the standard way to proceed for training and testing in the PTB corpus is to use section 2-21 to train the parser model, and then test it by feeding the parser with the section 23, and then, calculate *precision* and *recall* against the trees of PTB in that section.

Over the last years it is standard to report the harmonic mean of *precision* and *recall*, defined as:

$$F_1 = \frac{2 * precision * recall}{precision + recall}$$

2.1.2 Dependency Structures

Despite a long and venerable tradition in descriptive linguistics, dependency grammar has until recently played a fairly marginal role both in theoretical linguistics and in natural language processing. The increasing interest in dependency-based representations in natural language parsing in recent years appears to be motivated both by the potential usefulness of bilexical relations in disambiguation and by the gains in efficiency that result from the more constrained parsing problem for these representations (Niv05). In contrast with the constituent parsing, the dependency approach does not need to be "lexicalized" to solve ambiguity because this model is already lexicalized. The first relevant work in dependency syntactic analysis was by Tesnière (Tes59). In his work, the author defines explicitly the notion of dependency syntax[1]:

> "The sentence is an organized whole, the constituent elements of which are words. [1.2] Every word that belongs to a sentence ceases by itself to be isolated as in the dictionary. Between the word and its neighbors, the mind perceives connections, the totality of which forms the structure of the sentence. [1.3] *The structural connections establish dependency relations between the words*. Each connection in principle unites a superior term and an inferior term. [2.1] The superior term receives the name governor. The inferior term receives the name subordinate. Thus, in the sentence Alfred parle [. . .], parle is the governor and Alfred the subordinate.[2.2]"

Although this work was very influential in linguistics, it was not taken into account in the area of computer sciences. The formalization of this syntactic approach had to wait until (Gai65). In his work, the author encoded dependency structures with CFG. He proved that his formalization of dependency grammar is weakly equivalent to CFG[2]. Maybe it was the reason to not pay attention to dependency parsing, because it was considered equivalent to phrase structure parsing.

2.1.2.1 Dependency Trees

Formally, a dependency tree is defined in terms of a directed graph $G = (V, E, L)$. Given a possibly empty set R of dependency types (arc labels), a dependency graph for a sentence $x = (x_1, \dots, x_n)$ is a labeled directed graph, where:

- $V = \{1, \dots, n, n+1\}$
- $E \subseteq V \times V$
- $L : E \to R$

The following list contains some terminology for a dependency tree T:

- Since arcs are used to represent dependency relations, we will say that i is the head and j is the dependent of the arc (i, j).

[1]Translation made by Nivre in (Niv05) of the original work (Tes59).
[2]It is true for projective dependency trees.

- As usual, we will use the notation $j \rightarrow i$ to mean that there is an arc connecting i and j (i.e., $(i, j) \in E)$.
- The notation $j \rightarrow_* i$ if there is a directed path connecting the node j with the node i.
- The function L assigns a dependency type (arc label) $r \in R$ to every arc $e \in E$. The set R is the set of type of dependency relation. The basic type of relation are for example, $SUBJECT$, $OBJECT$ and $MODIFIER$. If the set R is empty, the tree is called unlabeled.

A dependency tree T is well-formed if and only if:

1. every word of index i in the sentence is a node $1 \leq i \leq n$ and there is a special node $n+1$, which does not correspond to any token of the sentence and which will always be the unique root of the dependency tree.
2. from every node $i \neq n+1$ in the graph there exists a path $i \rightarrow_* n+1$. It implies T is connected (Connectedness).
3. the tree is acyclic $\neg(j \rightarrow i \wedge i \rightarrow_* j)$
4. projectivity: $\forall : i, j : \ (i \rightarrow j \vee j \rightarrow i) \Rightarrow (\forall k, n : i < k < j < n : \neg(k \rightarrow n) \wedge \neg(n \rightarrow k))$. It is the same to say that there are no crosses arcs in T.

It is important to remark that the dependency trees that we used in this work are unlabeled. This decision was taken not only because Bilexical Grammar does not support labeled dependencies, but also due to the high computational cost of using optimization algorithms. Consequently, the underlying model to be used in an optimization algorithm should be as less complex as possible. Moreover, in unsupervised dependency parsing in general the dependency models are unlabeled (NHN$^+$07, CGS08, CS09, HJM09, SAJ10a, GGG$^+$10, BC10, WZ10).

2.1.2.2 Grammar and Treebank Dependency Parsing

As for natural language parsing in general, the first attempt to dependency parsing using a treebank was implemented by using a grammar and a corpus to induce a probabilistic model for disambiguation. Thus, in (CCCC92) the authors essentially use a PCFG model, where the CFG is restricted to be equivalent to a Gaifman (Gai65) type dependency grammar. They used an artificial corpus of POS tags.

They report experiments trying to induce such a probabilistic grammar using unsupervised learning on an artificially created corpus and using an inside-outside algorithm to induce the PCFG's probabilities. In those years there was not a dependency corpus available. The results obtained were very poor. After the PTB was created, dependency corpus started to be available. It is due to the fact that it is possible to transform a constituent corpus into a dependency one (Mag95b, Col97).

A few years later, Eisner (Eis96, Eis97) defined several probabilistic models for dependency parsing and evaluated them using supervised learning with data from the Wall Street Journal section of the Penn Treebank. In that work, Eisner showed how other models of dependencies in the literature (AB96, Col96) can be expressed under the general notion of a Probabilistic Bilexical Grammar (PBG)[1].

Formally, a PBG B is defined as a 3-uple $(C, \{r_c\}_{c \in C}, \{l_c\}_{c \in C},)$ where:

- C is a set of words tags, C contains a distinguished symbol $ROOT$.
- For each tag $c \in C$, l_c and r_c there are two probabilistic automata with start symbols S_{l_c} and S_{r_c} respectively.
- The probability of a dependency tree y under the PBG G is defined as the sum over all word tokens c in y, of the weight with which l_c accepts cs sequence of left children plus the probability with which r_c accepts cs sequence of right children.

Where a probabilistic automaton $A =< M, P >$ is defined as:

- M a deterministic finite automaton (DFA) and P is a probability model, that assigns probabilities to its transitions.

- The sum of the probabilities P of all transitions for a symbol c outgoing from state a is equal to 1.

- To each accepting path through A a probability is assigned, namely the product of all arc probabilities on the path.

- The probability of string x being accepted by A is the sum of the probabilities of its accepting paths.

[1]In the original definition of Eisner he calls it Weighted Bilexical Grammar (WBG), because the sum of all elements that belong to the language of a WBG is not necessarily 1. We use this abuse of notation as well as in the case of PCFG.

Although the definition of C is a set of words, the experiments made by Eisner used C as a set of POS tags.

To parse with the PBG, Eisner in (Eis00) showed how to transform the PBG in a PCFG, and uses a CYK algorithm to parse it in cubic time. A PBG can be seen as a PCFG $(S_{ROOT}, C, \bar{C} \cup (\bigcup_{c \in C} NTerm(G_{l_c}) \cup NTerm(G_{r_c})), R \cup (\bigcup_{c \in C} Rules(G_{l_c}) \cup Rules(G_{r_c})), P)$ where:

- S_{ROOT} is a new starting symbol associated with the Starting symbol of the PRG $G_{l_{ROOT}}$.
- C is the same set as in the PBG definition, which are considered the terminal symbols.
- $\bar{C}$ a set of non-terminal, defined : $\bar{c} \in \bar{C}$ iff $c \in C$.
- Given G_{l_c} and G_{r_c} two probabilistic regular grammars PRG that accept the same language that the automata l_c and r_c respectively, then $\bar{c} \rightarrow S_{G_l} c S_{G_r} \in R$, where S_{G_l} and S_{G_r} are the starting symbols of the regular grammars. Also the rules from the regular grammars belong to R. For each c $S_{ROOT} \rightarrow \bar{c} \in R$.
- The probability P of each rule is $\bar{c} \rightarrow S_{G_l} c S_{G_r} \in R$, and the probabilities of the rules from regular grammars are kept.

As we can see, the dependencies modeled in Eisner's work are unlabeled. In this thesis, as we based the representation in a PBG, the dependencies are unlabeled too and we will not give details of labeled dependencies approaches. For details about other dependency approaches that are not based on PCFG we refer to (Sam00, Niv04). (Niv05) is a more in depth coverage of the topics omitted.

2.1.2.3 Evaluation

The first NLP works using dependency structures (CCCC92, AB96, Eis00) were evaluated using bracketing precision and recall, as in the phrase structure parsing. As it is formally explained in (Kle04), the procedure is to transform the dependencies into constituent trees and then calculate the measures for phrase structure parsing.

In more recent works (Seg05, CGS08, HJM09, CS09, SAJ10a) the measures used are directed and undirected precision. Given a gold treebank GT of dependency trees and a set of parsed tree TP for the same sentences, the measures are defined as follows for each trees t_G and t_P in TG and TP respectively, with $YIELD(t_P) = YIELD(t_G)$:

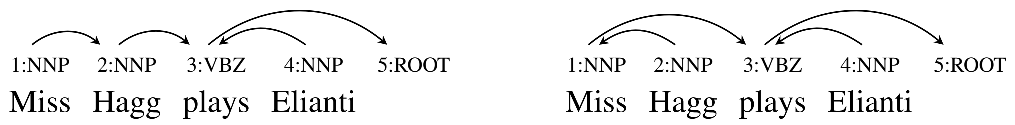


Figure 2.6: Two possible dependency trees for the PTB tree from `wsj_0297.mrg`.

- directed: number of arcs in t_P with the same direction and the same origin and destination that in t_G divided by the total number of arcs in t_G.
- undirected: as the directed but not taking into account the direction of the arcs.

The measures for the whole treebank PT against GT are calculated as the average of individual measures in each tree. For example considering the trees in Figure 2.6, suppose that the left tree is the GT and the right one PT, then the directed measure is $\frac{2}{4} = 0.5$ while the undirected measure is $\frac{3}{4} = 0.75$.

In dependency parsing, there exist two common baselines of evaluation, left attach and right attach. Right attach, means a treebank built with all the dependencies pointing to the right word in the sentence. That is for a sentence $X = x_1 \dots x_n$ the dependencies of a right attach tree are: $E = \{(1, 2) \dots (n, n+1)\}$. In an analogous way the left attach tree can be defined.

2.1.2.4 Treebanks

For English the PTB transformed to dependencies is used. But there exists corpora for other languages such as German (BDH$^+$02) and Czech (BHHH01) which are originally dependency corpora, that is, they have dependencies annotated. In the last five years, the standard dependencies corpora in dependency parsing are the ones used for the multilingual parsing competition “the ConNLL-X special task on parsing” (BM06). In that competition 13 different languages were used: Arabic, Bulgarian, Chinese, Czech, Danish, Dutch, German, Japanese, Portuguese, Slovene, Spanish, Swedish and Turkish. In our work on unsupervised parsing we used those corpora to train and evaluate our model.

2.1.3 From Constituent to Dependency and Viceversa

If we think of constituent and dependency grammars as two formalisms to describe underlying structures of the sentences, we can hope for a way to translate annotations from one formalism to the other. Given the high cost of constructing treebanks, automatic translators are desirable. For the dependency to constituent part, we notice constituents are formed by words and their dependents. Hence the algorithm should make the constituent structure explicit and provide the proper labels for the constituents.

For both parts, a more in depth coverage can be found in (UK04).

In order to translate a constituent structure to a dependency one, we must find among the words of a constituent which of them is the constituent head. These head-finding algorithms are based in sets of rules defined by linguistic theories. The first of these sets of rules was described in (Mag95b), and later works such as (Bik04a, Col97, KM02) use variants of that rule set. We use (Col97) as the current standard.

In every case, the rule sets were built by hand and not automatically optimized, as such they may be suboptimal for statistical parsers.

2.1.4 Some Models

In this section we provide a brief description of Collins' parser and the Factored Stanford Parser. Both parsers were used to evaluate the techniques implemented in Chapters 3 and 4. Also, the DMV parser (Dependency Model with Valence) is described, since it is a standard and baseline model in unsupervised dependency parsing and it is used to evaluate our work described in Chapter 5.

2.1.4.1 Collins' Parser

The Collins' parser is one of most accurate models for phrase structure parsing. Although the parser operates bottom-up, based on CKY algorithm, the probability and the rules of the grammar are defined in a top-down way. Every non-terminal label in every tree is lexicalized: the label is augmented to include a unique head word that the node dominates. The lexicalized PCFG that lies behind this model has rules of the form:

$$P \rightarrow L_n \ldots L_1 H R_1 \ldots R_m$$

where P, L_i ,R_i and H are all lexicalized non-terminals, and P inherits its lexical head from its distinguished head child H. In this generative model, first P is generated, then its head-child H, then each of the left- and right-modifying non-terminals are generated from the head outward. The modifying non-terminals

L_i and R_i are generated conditioning on P and H, as well as a distance metric (based on what material intervenes between the currently-generated modifying non-terminal and H). The process works recursively, treating each newly-generated modifier as a parent and then generating its head and modifier children; the process terminates when (lexicalized) pre-terminals[1] are generated. The results that this parser achieved were state-of-the-art with 88.63 of recall and 88.53 in precision. For more details about Collins parser, the reader can find a more in detail explanation in (Bik04c).

2.1.4.2 Stanford Parser

The version of the Stanford parser used to report the experiments in Chapter 4 is the parser described in (KM03a). In that work, the authors use a PCFG unlexicalized grammar. They use a similar structure that a head driven grammar, but instead of words the POS was percolated in the non-terminals of the grammar. They also used a split of POS according with the parent POS tag. They obtained a result close to the state-of-the-arts with $F_1 = 86.36\%$ using an unlexicalized model.

2.1.4.3 DMV Parser

This unsupervised dependency parser, the Dependency Model with Valence (DMV) (KM04) which was the first to beat the simple right-branching[2] baseline uses a probabilistic grammar for unlabeled dependency structures. It is defined as follows: the root of the sentence is first generated, and then each head recursively generates its right and left dependents. The underling grammar learned by DMV model can be seen as a PBG, where each automaton associated with a POS has a fixed structure, that we will describe in more detail in Chapter 5. The probabilistic model of automata is defined by two probabilities:P_{STOP} and P_{ATTACH}. $P_{STOP}(dir, h, adj)$ determines the probability to stop generating arguments, and it is conditioned by 3 arguments: the head h, the direction dir ((L)eft or (R)ight) and adjacency adj (whether the head already has dependents ((Y)es) in direction dir or not ((N)o)). $P_{ATTACH}(arg|h, dir)$ determines the probability to attach the dependent arg to the head h with direction dir. For the estimation of the probabilities, the optimization method EM algorithm is used, such algorithm will be explained in detail in the next Section. The performance obtained was 43.2 and 63.7 in directed and undirected

[1]The pre-terminals are usually the POS tags.

[2]The right attach performance of PTB for sentences with 10 words or less are 33.6 and 56.7 of directed and undirected accuracy respectively.

accuracy respectively, evaluated for English sentences from PTB which contain 10 words or less, ignoring punctuation marks.

2.2 Optimization Techniques

In this section we describe two different optimization techniques: Genetic Algorithms (GA) and the Expectation Maximization Algorithm (EM). For GA we used (Whi94, Mar04) as reference to write this section. The EM algorithm explanation is based on the contents of Prescher's work in (Pre03).

2.2.1 Genetic Algorithms

Genetic Algorithms work by analogy with evolution in biology. Starting with an initial population of solutions to a given problem, a new generation is created by combining and/or modifying the best members of the population. The process is then iterated until some adequate criterion for termination is reached.

Taking the analogy further, we assume that we have a genetic description of candidate solutions. That is, a solution can be represented as a vector of roughly independent parameters which will be modified from each generation to the next. We also need a practical method for evaluating the goodness of candidate solutions and hence select the best members of the population.

A generic implementation of a genetic algorithm is shown in Algorithm 4. We shall briefly comment on each of the stages of the algorithm.

Initialization: a set of candidate solutions can be generated completely at random, manually selecting candidates in promising areas of the search space, or using some hybrid method.

The phases of evaluation, selection and reproduction are not necessarily performed sequentially, but often they are. For the evaluation stage, a good fitness function should be fast to evaluate and able to discriminate among the candidate solutions. It is common to sacrifice adequacy of the evaluation against efficiency of the implementation, allowing for approximations to the “real” objective function to be used.

The selection of the candidates to pass their genes to the next generation has been implemented in various ways. One can simply choose the k best solutions, do so with some bias towards better solutions, take into account the “ages” of the individuals, use some sort of threshold, human picking, and so on.

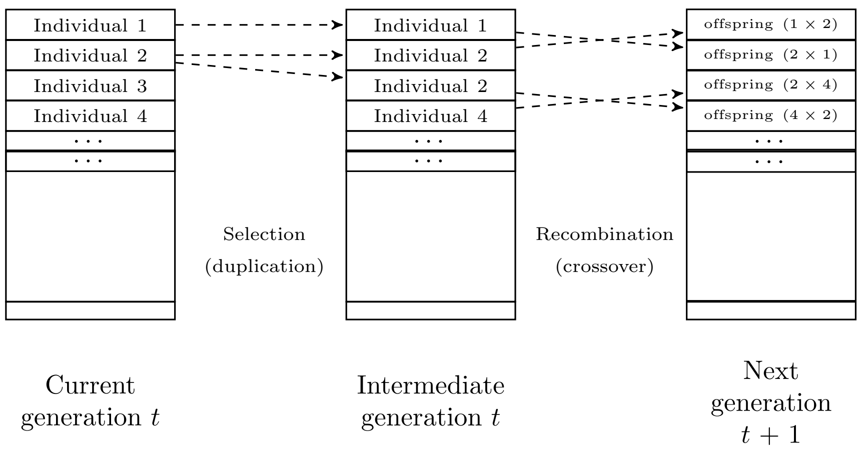


Figure 2.7: Creation of a generation in a Genetic Algorithm.

The phase of reproduction is where the potential for improvement of the solution arises. The main two mechanisms for modifying solutions are mutation and crossover.

Mutation operates by changing the values for one or more chromosomes of the parent individual. Its purpose is to maintain diversity among the populations and so give the algorithm a chance to explore different regions of the search space and avoid getting stuck at a local optimum.

Crossover involves the construction of a solution by combining two (or more) "parent" solutions. The logic behind it is to accelerate the propagation of "good" parts of the solutions. For example, a quick frog can mate a green frog and then green and quick frogs will appear in the population. (Slower and not-so-green frogs are not expected to survive; and if they do, they are lucky frogs who could be kissed by zoophilic princess.)

The criterion for termination may vary. One can choose among different possibilities such as running the algorithm for a fixed amount of time or iterations, stopping when an acceptable solution is found, when a number of rounds have passed without improvements, and so on.

2.2.1.1 The Selection Process

It is helpful to view the execution of a GA as a two stage process. It starts with the **current population**. First a selection stage is applied to the population to create an **intermediate population**. Second, the recombination and mutation are applied to

```
Data:
    • $f$ the evaluation function
    • $CS$ a constant of stop criteria
    • $n$ the size of population
    • $r$ the probability for applying crossover
    • $m$ the probability of apply mutation
Result: $p_{CS}$ the best individual according with the evaluation
        function.
1  for $i = 1$ to $n$ do
2      add a random $p$ in $P$; calculate $f(p)$
3  end
4  ;
5  $AVG\_Q\_PREV := \infty$; $AVG\_Q\_NEW := AVG(f, P)$;
6  while $|AVG\_Q\_PREV - AVG\_Q\_NEW| < CS$ do
       /* Create a intermediate population $P_I$ :
          */
7      $Selection$:
         for all $p_i \in P$ calculate $FITNESS(p_i) = \frac{f(p_i)}{\sum_{p_j \in P} f(p_j)}$
         select $(1 - r) * n$ ind. from $P$, proportional to $FITNESS()$;
         add those members to $P_I$ ;
8      $Crossover$:
         select $\frac{r*p}{2}$ ind. from $P$, proportional to $FITNESS()$;
         for each pair $(p_1, p_2)$ apply cross over;
         add the new children to $P_I$
                    chose an $m$ percent of members from $P_I$;
9      $Mutation$:  apply mutation to the chosen elements;
                    add the new children to $P_I$
10     $P := P_I$;
                   calculate $f$ for each element in $P_I$;
11     $Evaluate$: $AVG\_Q\_PREV := AVG\_Q\_NEW$;
                   $AVG\_Q\_NEW := AVG(P, f)$
12 end
13 return $\text{argmax}_f(P)$;
```

Algorithm 4: The algorithm $GA(f, CS, n, r, m)$.

the intermediate population to create the **next population**. This two state process is completed and it forms a generation in a GA execution. There are many ways to do the selection process to build the intermediate population, the following are the most used criteria:

- Elitist selection: The most fit members of each generation are guaranteed to be selected. (Most GAs do not use pure elitism, but instead use a modified form where the single best, or a few of the best, individuals from each generation are copied into the next generation just in case nothing better turns up.)
- Fitness-proportionate selection: More fit individuals are more likely, but not certain, to be selected.
- Roulette-wheel selection: A form of fitness-proportionate selection in which the chance of an individual of being selected is proportional to the amount by which its fitness is greater or less than its competitors' fitness. (Conceptually, this can be represented as a game of roulette - each individual gets a slice of the wheel, but more fit ones get larger slices than less fit ones. The wheel is then spun, and whichever individual "owns" the section on which it lands each time is chosen.)
- Scaling selection: As the average fitness of the population increases, the strength of the selective pressure also increases and the fitness function becomes more discriminating. This method can be helpful in making the best selection later on when all individuals have relatively high fitness and only small differences in fitness distinguish one from another.
- Tournament selection: Subgroups of individuals are chosen from the larger population, and members of each subgroup compete against each other. Only one individual from each subgroup is chosen to reproduce.

2.2.1.2 Making Changes

After the selection process occur, the intermediate population have been created, and the recombination is carried out followed by the mutation, as can be seen in Figure 2.7. After this process the next generation is created. The recombination process is implemented by applying to randomly paired individuals with a probability p_c[1]. The crossover entails choosing two individuals with the probability p_c

[1]The population should be shuffled by the selection process.

to swap segments of their code, producing artificial "offspring" that are combinations of their parents. This process is intended to simulate the analogous process of recombination that occurs to chromosomes during sexual reproduction. Common forms of crossover include single-point crossover, in which a point of exchange is set at a random location in the two individuals' genomes, and one individual contributes all its code from before that point and the other contributes all its code from after that point to produce an offspring. For example, suppose that in a GA with individuals encoded in binary strings of length 10, and we want to crossover the two strings: 1111111111 and 0000000000, suppose that the selected point is 3, then, the new two individuals generated are 0001111111 and 1110000000. Other common method is the uniform crossover, in which the value at any given location in the offspring's genome is either the value of one parent's genome at that location or the value of the other parent's genome at that location, chosen with fifty-fifty probability.

After the recombination the mutation procedure is performed. In this process the individuals must be randomly altered in hopes of improving their fitness for the next generation according with a probability p_m. For example, in implementations of GA individuals using binary strings, the mutation is usually performed by flipping the value of some bits of the individual, and in other cases the implementation of some bits of the individuals are randomly generated.

Although p_c and p_m are estimated by running the GA and analyzing the fitness evolution across a few generations, usually p_c is close to 0.6 and p_m is around 0.01.

2.2.1.3 Why (and when) do Genetic Algorithms work?

Genetic Algorithms are robust enough to be able to solve a wide variety of problems, nevertheless there are some problems more suitable than others to be solved by this technique. Some characteristics that the problem must satisfy are:

- The problem and its solutions must to be encoded as a combination of multiple parameters. The good news here is that these parameters do not have to be necessarily homogeneous but may include real-valued parameters mixed with discrete ones.

- As we evaluate the fitness of solutions repeatedly, the function that does so must be fast and be able to order the goodness of candidate solutions; a yes/no valued function does not help too much.

- GAs are best when we try to optimize a set of interdependent parameters simultaneously. Better methods exist if we can solve the problem optimizing the parameters sequentially (linear programming is an example).

The following are the properties of GAs that were taken in consideration to solve the optimization problems presented in our work.

- The first and very important point is that genetic algorithms are intrinsically parallel. The fitness function can be calculated over each individual in parallel and so can the crossover and mutation be implemented.
- GAs explore the solution space in different regions and diverse directions at the same time. Moreover, they work by quickly combining multiple "good" solutions for parts of the problem in a globally better solution. Other algorithms usually work by trying to refine a single solution at a time and they have a tendency to get stuck at local minimal. Since the population in GAs is a set of solutions, the diversity among the population amounts to this parallel exploration of the search space.
- Genetic Algorithms excel at working with multi-parameter, mixed-valued solutions. They are fast to find promising regions of the search space that can be seeded to other methods for further refinement.
- The knowledge about the solution space of the problem that is implicitly put in the algorithm is limited as that solution must admit genetic descriptions. The down side is that hard-wiring domain knowledge may be difficult, but the other side of the coin is that we can gain insight about an unknown solution space from the results of a Genetic Algorithm.

2.2.2 Expectation Maximization

As saw earlier in this chapter, in many cases of NLP research a common problem to solve is to receive a corpus and try to find the unknown probability distribution which generates it. As we have previously seen the MLE (Maximum Likelihood Estimation) is a good estimator for a probability distribution when we have available a large corpus of annotated trees. However, if such corpus is not available we cannot apply MLE directly. The Expectation Maximization algorithm (EM algorithm) is designed to estimate a probability distribution that maximises the likelihood when not all the parameters are necessary to estimate it. For example, this is the case of

PCFG parser which does not have a large corpus of annotated sentences. The next section introduces some definitions necessary to describe the EM-Algorithm.

2.2.2.1 Some definitions

We will start formalizing the concept of corpus which is defined as a real-valued function $f : Y \rightarrow Real$, with Y a numerable set such that:

- $f(y) \geq 0$ for all $y \in Y$
- for each $y \in Y$ the value $f(y)$ returns the frequency of the element y
- the size of the corpus is defined as $|f| = \sum_{y \in Y} f(y)$
- the corpus is finite and non-empty if $0 < |f| < \infty$

We now give the formal definition of the probability of f, a non-empty finite corpus over Y, being generated by a probability distribution. Let M be the set of all possibles probabilities that can be defined for Y and let $p \in M$, then:

- the probability of the corpus based on the probability p is a function L such that: $L(f, p) = \prod_{y \in Y,\ f(y) \neq 0} p(y)^{f(y)}$
- a probability function $\hat{p} \in M$ is called a Maximum Likelihood Estimator of M on f, iff the corpus f have the probability $L(f, \hat{p}) = \text{argmax}_{p \in M} L(f, p)$
- as it is shown in (Pre03), when f is finite and non-empty, then the MLE is unique and it is the same that the probability function calculated by using the relative frequency
$$\hat{p}(x) = \frac{f(x)}{|f|}$$

We now formalize the notion of an analyzer. A symbolic analyzer is a device that assigns to each incomplete data type a set of analysis, each analysis being a complete data type. Let X and Y two non-empty numerable sets. A function $A : Y \rightarrow 2^X$ is called:

- symbolic analyzer if the set of analysis $A(y) \subseteq X$ are pair-wise disjoint, and the union is complete $X = \bigcup_{y \in Y} A(y)$

- the set Y is called incomplete-data type and X are called the incomplete-data set. As it is shown in (Pre03) the analysis A form a partition of the complete-data set X. Therefore, for each $x \in X$ exists a unique $y \in Y$, usually named the $yield$ of x, such that x is an analysis of y:

For example, if we are working with the CFG as PSG, like earlier in this Chapter, the incomplete-data type Y is a corpus of sentences and the phrase structure trees are the complete data type X. Tipically, the symbolic analyzer is the parser for the CFG, where the parser assign to each sentence a set of the possible phrase structures trees for this sentence.

As it was done for parsers, we now will define the statistical version of analyzers.

- A pair $< A, p >$ where $A : Y \to 2^X$ is an symbolic analizer and p a probabiliistic distribution over X, is called a statistical analyzer.
- The statistical analyzer is used to induce probabilities for the incomplete-data elements $y \in Y$, such that $p(y) = \sum_{x \in A(y)} p(x)$.
- To solve the ambiguity, given that for each element of x there may be many elements associated by $A(x)$, we use conditional probablities defined as: $p(x|y) = \frac{p(x)}{p(y)}$ where $y = yield(x)$.
- It is easy to check that $p(y)$ and $p(x|y)$ are well-formed probability distributions, you can find a proof by Prescher in (Pre03) page 10.

2.2.2.2 The algorithm

The input of an EM algorithm is an incomplete data set together with a symbolic analyzer. The idea of EM-algorithm is to induce a probabilistic model for the symbolic analyzer, which is capable of "solving" the ambiguity present in the analyzer. This ambiguity is present because for each element $y \in Y$, A associates a set of elements on X. The induced probability model p will decide which of the analysis of each element is more likely. This optimization algorithm performs a sequence of runs. In each run the algorithm performs a step of "Expectation" followed by the "Maximization" step. In the E-step the symbolic analyzer is paired with the probability model p_{i-1} calculated in the previous loop, this combination solves the

Data:

- the symbolic analyzer $A : Y \rightarrow 2^X$,
- a non empty incomplete-data corpus, i.e a frequency function $f : Y \rightarrow Real$, such that $\forall y \in Y :\ f(y) \geq 0$ and $0 < |f| < \infty$
- a randomly start probability for the incomplete data type p_0 such that $< A, p_0 >$ form the initial statistical analyzer

Result: $p[i]$ the probability distribution for the incomplete data corpus Y

```
1 for $i = 1$ to $MAX\_ITER$ do
2     $q = p[i-1]$;
3     E-Step : compute $f_q : X \rightarrow Real$:
          $$f_q(x) = f(y) * q(x|yield(x))$$
      ;
4     M-Step : compute the MLE $\hat{p}$ of $M$ (probability models over $Y$) on $f_q$:
          $$L(f_q, \hat{p}) = \operatorname*{argmax}_{p \in M} L(f_q, p) = \prod_{x \in X,\ f_q \neq 0} p(x)^{f_q(x)}$$
      ;
5     $p[i] = \hat{p}$;
6 end
7 return $p[i]$;
```

Algorithm 5: The EM algorithm.

$f(y)$	y
10	$y_1 =$ ”$n\ p\ p$”
5	$y_2 =$ ”$n\ p$”

Table 2.5: An example of the input frequency for the example grammar from Table 2.6.

S → N P	
P → P N	P → p
P → P P	
N → N P	N → n

S → N P	1.0		
P → P N	0.33	P → p	0.33
P → P P	0.33		
N → N P	0.5	N → n	0.5

Table 2.6: Left: A CFG grammar G. Right: the left CFG with uniform probabilities in its rules.

ambiguity present in A. The M-step calculates a MLE $\hat{p}$ based in the statistical analyzer $< A, p_{i-1} >$. In Algorithm 5 a pseudo code of EM-algorithm is shown.

As it can be observed, this algorithm ensures that the likelihood of each $p_1, \ldots, p_n$ as given in each step of the EM-algorithm and calculated over the complete-data set Y is monotonic increasing. That is:

$$L(f, p_0) \leq \ldots L(f, p_n) \leq \ldots$$

.

Next, we introduce an example of how to use the EM-algorithm to induce the probabilities of a given PCFG. Suppose that we have the following CFG $G = (S, \{N, P\}, \{n, p\}, R)$ as shown at the left side of Table 2.6. In this example, our corpus of sentences is $Y = \{y_1, y_2\}$, the input frequency $f(y)$ is shown in Table 2.5.

The input Treebank $X = \{x_1, x_2, x_3\}$ is the one shown in Figure 2.8. For the initial probability model for G we will use an uniform distribution for each left hand side of the rules p_0.We show at the right side of Table 2.6 the PCFG $< G, p_0 >$.

The first step of the EM-algorithm is performed as follows: the E-Step will calculate the frequency of f_{T_q}, which is the frequency of the treebank, based on $q = p_0$. The initial probability p_0 for the sets X and Y is generated as follows:

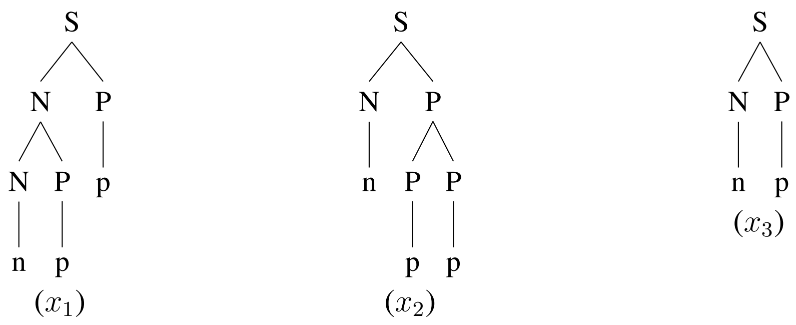


Figure 2.8: A treebank of trees.

$$\begin{aligned}
p_0(x_1) &= \textstyle\prod_{r\in Rules(G)} p(r)^{f_r(x_1)} \text{ ($f_r(x)$ is the \#occurrence of rule r in the tree x)}\\
&= p(s \to N\ P) * p(N \to N\ P) * p(N \to n) * p(P \to P) * p(P \to p)\\
&= 1 * 0.5 * 0.5 * 0.33 * 0.33\\
&= 0.027225\\
p_0(y_1) &= \textstyle\sum_{yield(x)=y_1} p(x)\\
&= p(x_1) + p(x_2)\\
&= 0.027225 + 0,017968\\
&= 0.045193
\end{aligned}$$

After calculating all elements we have induced all values of p_0:

x	$p_0(x)$
x_1	0.027225
x_2	$0,017968$
x_3	$0,165$

y	$p_0(y)$
y_1	0.045193
y_2	$0,165$

Now using the probability $q = p_0$ we can calculate the f_{T_q} as in line (3) of the EM-algorithm.

$$\begin{aligned}
f_{T_q} &= f(yield(x_1)) * q(x_1|yield(x_1))\\
&= f(y_1) * q(x_1|y_1)\\
&= f(y_1) * \tfrac{q(x_1)}{q(y_1)}\\
&= 10 * \tfrac{0.027225}{0.045193}\\
&= 6.02
\end{aligned}$$

finally the complete values of $f_{T_q}(x)$ are:

x	$f_{T_q}(x)$
x_1	6
x_2	4
x_2	5

In the M-step of the EM-algorithm used to estimate the underlying distribution of a PCFG, the idea is to induce the probability $p_{T_q}(x)$ for the grammar G from the frequency corpus $f_{T_q}(x)$ calculated in the E-step. We use the MLE to calculate the probabilities of the rules. The frecuency of a given rule r in a corpus $f_{T_q}(x)$ is:

$$f(r) = \sum_{x \in X} f_{T_q}(x) * f_r(x)$$

For the rules in our example we have:

$$\begin{aligned}
f(P \to P\ N) &= \textstyle\sum_{x \in X} f_{T_q}(x) * f_{P \to P\ N}(x) \\
&= f_{T_q}(x_1) * f_{P \to P\ N}(x_1) + f_{T_q}(x_2) * f_{P \to P\ N}(x_2) + f_{T_q}(x_3) * f_{P \to P\ N}(x_3) \\
&= 6 * 0 + 4 * 0 + 5 * 0 \\
&= 0 \\
f(N \to N\ P) &= \textstyle\sum_{x \in X} f_{T_q}(x) * f_{N \to N\ P}(x) \\
&= f_{T_q}(x_1) * f_{N \to N\ P}(x_1) + f_{T_q}(x_2) * f_{N \to N\ P}(x_2) + f_{T_q}(x_3) * f_{N \to N\ P}(x_3) \\
&= 6 * 1 + 4 * 0 + 5 * 0 \\
&= 6 \\
f(P \to P\ P) &= \textstyle\sum_{x \in X} f_{T_q}(x) * f_{P \to P\ P}(x) \\
&= f_{T_q}(x_1) * f_{P \to P\ P}(x_1) + f_{T_q}(x_2) * f_{P \to P\ P}(x_2) + f_{T_q}(x_3) * f_{P \to P\ P}(x_3) \\
&= 6 * 0 + 4 * 1 + 5 * 0 \\
&= 4 \\
f(P \to P) &= \textstyle\sum_{x \in X} f_{T_q}(x) * f_{P \to P}(x) \\
&= f_{T_q}(x_1) * f_{P \to P}(x_1) + f_{T_q}(x_2) * f_{P \to P}(x_2) + f_{T_q}(x_3) * f_{P \to P}(x_3) \\
&= 6 * 2 + 4 * 2 + 5 * 2 \\
&= 30 \\
f(N \to N) &= \textstyle\sum_{x \in X} f_{T_q}(x) * f_{N \to N}(x) \\
&= f_{T_q}(x_1) * f_{N \to N}(x_1) + f_{T_q}(x_2) * f_{N \to N}(x_2) + f_{T_q}(x_3) * f_{N \to N}(x_3) \\
&= 6 * 1 + 4 * 1 + 5 * 1 \\
&= 15
\end{aligned}$$

Using this frequency the probability for a rule $r = s \to t$ can be calculated as:

$$p_{T_q}(x) = \frac{f(r)}{\sum_{r_i = s \to * : r_i \in Rules(G)} f(r_i)}$$

The probabilities of rules of the grammar G after the first step of the EM-algorithm are:

S $\rightarrow$ N P	1.0		
P $\rightarrow$ P N	0	P $\rightarrow$ p	0.833
P $\rightarrow$ P P	0.1667		
N $\rightarrow$ N P	0.285	N $\rightarrow$ n	0.715

As a consequence of this calculation, $p_1 = p_{T_q}$ and the next iteration of the probabilistic model $p_2 \ldots$ can be performed using these values as input.

Chapter 3

Joining automata to optimize split of POS tags

The work described in this chapter is based on the article published in (DIL08). In this research task we introduce a technique for inducing a refinement of the set of part of speech tags related to verbs. Each of the resulting sets defines a new POS tag. We try out the resulting tag set in a state-of-the art phrase structure parser and we show that the induced part of speech tags significantly improves the accuracy of the parser.

3.1 Introduction

A part-of-speech (POS) tag is a linguistic category of words that are characterized by their particular syntactic behaviors. Originally, POS tags were defined in eight basic categories, derived from Latin grammar. However, for NLP research this categorization was not descriptive enough to capture the syntactic information necessary for syntactical structure processing. For example, the developers of the PTB defined more than thirty grammatical categories.These POS tags are usually defined within a syntactic theory. Supervised algorithms for parsing use POS tags as they are defined.

The importance of POS definitions for parsing lies in the fact that the grammars inferred to build the parser uses these categories to define the grammar rules. Nevertheless, a given definition of POS tags may not be the best for supervised algorithms for parsing; indeed when POS tag sets are defined they are not intended for this particular research purpose. It is in principle possible that words might be grouped differently in order to improve parsing performance.

Our main research question is: Can we redefine the set of POS tags so that when a state-of-the-art parser is trained using this new set its performance improves? We answer this question by presenting an algorithm that induces sets of POS tags capable of improving state-of-the-art-parsing performance. We show that our POS tag sets improve parsing by means of encoding some additional linguistic information into the new set of tags, which is clearly useful for the parsing model.

We extract information from dependency trees based on Bilexical Grammars (Eis97). As we described in Section 2.1.2.2, in Bilexical Grammars there are two automata for each word in the lexicon. These automata model, respectively, right and left dependents. We cluster words whose automata are "similar", and we treat each cluster as a new POS tag.

We design a procedure that implements this simple idea; a procedure that aims at finding the best possible POS tag set clustering words whose automata show similar behaviors. The procedure is defined as an optimization problem. As usual in optimization problems, we define the quality measure it has to optimize, its search space, and strategy it should follow to find the optimal POS tag set among all possible tag sets.

The *search space* also defines the type of information that will be codified into the POS tags. Different syntactic features are used to generate different search spaces and, consequently, different resulting POS tag sets. The *quality measure* for a tag set is computed by tagging a dependency tree-bank using the POS tag to be evaluated, inducing a bilexical grammar and evaluating the grammar's quality, so that the quality measure for POS tags is evaluated using quality measures for bilexical grammars. Finally, the *strategy* for traversing the search space is implemented using Genetic Algorithms.

The set of new POS tags can be used to retag a phrase structure corpus. In our case, we retag the Penn Treebank (PTB) (MS93) to test our new sets of POS tags in a phrase structure setting. POS tag sets are evaluated using Collins parser (Col97) by means of Bikel's implementation (Bik04b). We add our new POS tags to the training material, we re-train the parser and we evaluate the parser performance, showing a significant improvement on parsing results.

The rest of the chapter is organized as follows. Section 3.2 gives an overview of different approaches to the same problem from the literature. Section 3.3 provides details on how to compute the quality measure, Section 3.4 explains how to build and traverse the search space, Section 3.5 explain how the new tag sets are used for training a phrase structure parser and it also reports the performance for the different tag sets we build. Section 3.6 hints some possible future directions. Finally, Section 3.7 present a Discussion of the placement of our work in the research area, and also concludes the Chapter.

3.2 Related Work

Two relatively recent approaches studied the use of automatic split of non-terminals to improve parsing performance. In (MMT05) the authors induce a parsing model by using a generative process which starts with a standard PCFG and splits each non-terminal in a fixed number of categories. The resulting model is a generative unlexicalized grammar named PCFG-LA. Using this parser they obtained a F_1 measure comparable to state of the art unlexicalized parsers. Similarly, in (PBK06) the authors induce a PCFG grammar using an automatic split and merge of non-terminal symbols to maximize the likelihood of training treebank. The hierarchical split and merge used also captures linguistic phenomena that used to be added manually in previous works. Furthermore, they obtain a lexicalized parser model with a performance comparable to the parser in (Cha00) and the grammar induced is significantly smaller. In contrast, our approach uses an automatic unlexicalized split only for pre-terminal symbols, and we use the new POS set to rewrite the training material for a given parser to improve its performance.

Rewriting the training material is an important aspect of our work that has been studied in the literature. For example, Mohri and Roark (MR06) present a technique that induces better performing PCFGs. They split and factorize *non-terminals* that have been detected as structural zeros in a given training material. Our approach differs from theirs in that we split only *pre-terminal* and that our splitting is based on syntactic behavior.

Klein and Manning (KM01) split POS tags related to verbs in order to detect constituents. They did this in the context of induction of rules for grammars. All categories used, such as transitive, intransitive, etc., were set in advance. In (SJ01) the authors use a fixed number of categories that are based on universal language rules to build an unsupervised POS tagger. In contrast to the latter two approaches, our approach induces all categories automatically and, moreover, the resulting categories are tested in phrase structure-grammars, providing a better way to assess the quality of the resulting tags.

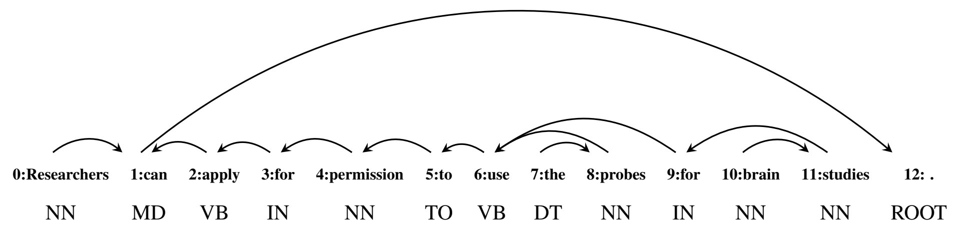


Figure 3.1: Tree extracted from the PTB, file `wsj_0297.mrg` and transformed to a dependency tree.

3.3 Quality Measure for Tag Sets

This section introduces the quality measure q that we used for evaluating and optimizing POS tag sets. This measure is defined using a further quality measure q' for a particular flavor of bilexical grammars. Briefly speaking, the quality q of a tag set C is computed by means of retagging a dependency tree-bank with C, inducing a bilexical grammar B from it, and computing a quality function q' on B.

This section shows how bilexical grammars are induced from dependency tree-banks and how the measure q is defined in terms of q'. It also discusses why we think q is a good measure of the quality of a tag set C.

This measure q' takes into account the quality of all automata that define the grammar B.

3.3.1 Induction of Bilexical Grammars

Bilexical grammars can be induced from a dependency tree-bank by inducing two automata for each tag in C. Once the tag set C is defined, the induction of a bilexical grammar is straightforward. The induction of Bilexical Grammars is carried out in a supervised fashion. Our training material comes from Sections 02–21 of the PTB. Trees are first transformed to dependency trees using Collins rules as implemented by Bikel's parser. All words in the PTB are removed and original POS tags are replaced by tags in a given tag set C. This means that for each candidate POS tag set C the training material has to be rewritten.

Once the training material reflects the tag set C, two bags T_L^c and T_R^c of *strings* for each tag c in C are extracted. An example illustrates the extraction procedure better: Figure 3.1 shows a dependency tree and Table 3.1 shows some of the bags of left and right dependents that are extracted. Left dependents are read from right to left, while right ones from left to right. In both cases, all strings in the bag showed in Table 3.1 start with the word w and end with $\#$. Note that in the example, the tag set C is the PTB tag set, and all sets of strings displayed in the table are strings extracted from the example tree only. In the actual setting, T_L^c and T_R^c are built joining strings coming from all trees in the tree-bank.

Once T_L^c and T_R^c are extracted, two probabilistic automata A_L^c and A_R^c are built. For this purpose, we use the *minimum discrimination information* (MDI) (TDdlH00) algorithm. The MDI algorithm receives as arguments a bag of strings and it outputs a probabilistic deterministic automata that accepts and generalizes over the input bag of strings. The MDI algorithm has a unique parameter $0 \leq \texttt{alpha} \leq 1$, which controls how the resulting automata generalizes the training material;with $\texttt{alpha} = 0$ the automata generates only the training material, while values close to 1 have the opposite behaviour, i.e. the automata over genealizes the training material. Such parameter is optimized during the grammar optimization phase as explained in Section 3.4. Since a bilexical grammar is defined through its automata, once all automata A_L^c and A_R^c, c in C are induced, the bilexical grammar associated to the tag set C is completely defined.

Word #	i	T_L^i	T_R^i
0	NN	{NN #}	{NN #}
1	MD	{MD NN #}	{MD VB #}
2	VB	{VB #}	{VB IN #}
3	IN	{IN #}	{IN NN #}
4	NN	{NN #}	{NN TO #}
5	TO	{TO #}	{TO VB #}
6	VB	{VB #}	{VB NN IN #}
7	DT	{DT #}	{DT #}
8	NN	{NN DT #}	{NN #}
9	IN	{IN #}	{IN NN #}
10	NN	{NN #}	{NN #}
11	NN	{NN NN #}	{NN #}
12	ROOT	{ROOT MD #}	{ROOT #}

Table 3.1: Bags of left and right dependents extracted from dependency tree in Figure 3.1. Left dependents are to be read from right to left. All displayed sets are singletons.

3.3.2 Quality Measure for Grammars

The measure q' for bilexical grammars is based on two quality measures for probabilistic automata (ILdR04). The first, called *test sample perplexity* (PP), is the *per symbol log-likelihood* of strings belonging to a test sample according to the distribution defined by the automaton. The minimal perplexity $\mathrm{PP} = 1$ is reached when the next symbol is always predicted with probability 1, while $\mathrm{PP} = |\Sigma|$ corresponds to uniformly guessing from an alphabet Σ of size $|\Sigma|$. The second measure is given by the number of *missed samples* (MS). A missed sample is a string in the test sample that the automaton fails to accept. One of such instance suffices to have PP undefined. Since an undefined value of PP only witnesses the presence of at least one MS we count the number of MS separately, and compute PP without considering MS. The test sample that is used to compute PP and MS comes from all trees in sections 00-01 of the PTB. These trees are transformed to dependency trees and they reflect tag sets C as the training material.

We can now define the measure q' for bilexical grammars. q' has two parts, one considering all automata related to right-hand side dependents, and one considering left-hand side. To simplify our exposition, we give the component referring to the right-hand side; the other component is obtained by replacing R in the superscripts with L.

Let $C = \{c_1, \ldots, c_n\}$ be a candidate tag set. Let $A_R^{c_i}$, $i = 1, \ldots, n$ be the automata induced as described previously. Let $\mathrm{PP}_R^{c_i}$ and $\mathrm{MS}_R^{c_i}$ be the values of PP and MS respectively for the automaton $A_R^{c_i}$. We combine all values of PP_i and MS_i to obtain a quality value for the whole grammar.

PP and MS values can not simply be summed up as the importance of an automaton is proportional to the number of times it is used in parsing, as a consequence we combine the different values of PP and MS using weights. Let's define $p_R^{c_i} = |T_R^{c_i}|/|T_R|$, where $i = 1, \ldots, n$; where T_R is union of all $T_R^{c_i}$; we view $p_R^{c_i}$ as the probability of using the automata $A_R^{c_i}$. Let $E[\mathrm{MS}_C^R]$ and $E[\mathrm{PP}_C^R]$ be the *expected value of* MS *and* PP *for a right automata*, defined as $E[\mathrm{MS}_C^R] = \sum_{i=1}^n p_R^{c_i}\mathrm{MS}_R^{c_i}$, and $E[\mathrm{PP}_C^R] = \sum_{i=1}^n p_R^{c_i}\mathrm{PP}_R^{c_i}$, respectively. Let $E[\mathrm{MS}_C^L]$ and $E[\mathrm{PP}_C^L]$ be the corresponding values for the left sides. The expected values depend on a tag set, hence the subscript C.

The quality measure q' for a bilexical grammar B is defined using $E[\mathrm{PP}_C^R]$, $E[\mathrm{MS}_C^R]$, $E[\mathrm{PP}_C^L]$ and $E[\mathrm{MS}_C^L]$. Formally, the function q'_{C_0} that we minimize for grammars is

$$q'_{C_0}(B) = \begin{cases} \|X\| + k & \text{if } E[\mathrm{PP}^R_C] > E[\mathrm{PP}^R_{C_0}] \\ \|X\| + k & \text{if } E[\mathrm{MS}^R_C] > E[\mathrm{MS}^R_{C_0}] \\ \|X\| + k & \text{if } E[\mathrm{PP}^L_C] > E[\mathrm{PP}^L_{C_0}] \\ \|X\| + k & \text{if } E[\mathrm{MS}^L_C] > E[\mathrm{MS}^L_{C_0}] \\ \|X\| & \text{otherwise,} \end{cases}$$

where

$$X{=}(E[\mathrm{PP}^R_C], E[\mathrm{MS}^R_C], E[\mathrm{PP}^L_C], E[\mathrm{MS}^L_C]),$$

$$\|(x_1, x_2, \ldots, x_n)\| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2},$$

k is a constant used to penalize configurations that we know are not part of the set of possible solutions, and C_0 is the set of POS tags defined by the PTB. Finally, the function $q(C)$ for a given candidate POS tag C is defined as $q'(B)$ where B is the bilexical grammar that can be induced using C as the tag set. Note that, q' and q are essentially the same function, they only differ on the type of the argument they take. Given that q uses C_0 as a referent, we can see q as a function that penalizes POS tag sets whose expected values of PP and MS are worse than those values obtained by the PTB tag set. Better values of MS and PP for a grammar mean that its automata capture better the regular language of dependents by producing most strings in the automata target languages with fewer levels of perplexity.

Another point of view on q comes from formal language theory: for a given tag c, the automata A^c_R and A^c_L model the probabilistic regular language of right and left dependents respectively. The idea behind q' is that these probabilistic languages might be better described as the disjoint union of several smaller probabilistic regular languages. The two measures (PP and MS) in which q is based, indirectly measure the diversity of languages that are associated to each POS tag. Such analysis was first introduce in (ILdR04). As we show in the next section, the optimization tries to detect which are these several languages and to define a new POS tag for each of them.

3.4 Building and Traversing the Search Space

The search space is built by means of 2 elements: a subset V of PTB POS tags and a function f called *feature*. V is the portion of the PTB tag set that we want to refine. f is a function that takes two arguments, a dependency tree t and a number i. The number i refers to the i-th node, from left to right, in the dependency tree t. Since words in the yield of t are in direct correspondence to its nodes, the index i also corresponds to the i-th word in the yield of t. A feature returns some information around the i-th node in the tree; they are meant to characterize the dependents a verb might take. Figure 3.2 lists a few features and it shows examples of features applied to the tree in Figure 3.1.

VerbAllDepth: the number of nodes labeled with {`VB,VBD,VBN,VBZ,MD,TO`} in the path that goes from node i to the ROOT node. $f(t, 6) = 3$.

VerbDepth: the number of nodes labeled with {`VB,VBD,VBN,VBZ,MD`} in the path that goes from node i to the ROOT node. $f(t, 6) = 2$.

VerbVBDepth: the number of nodes labeled with {`VB,VBD,VBN,VBZ`} in the path that goes from node i to the ROOT node. $f(t, 6) = 1$.

NumSib: the number of sibling. $f(t, 2) = 1$.

NumDep: the number of dependents. $f(t, 6) = 2$.

NumChanges: the number of times the label changes in consecutive nodes in the path that goes from node i to the ROOT node. $f(t, 10) = 7$.

Depth: The length of the path that goes from node i to the ROOT node. $f(t, 6) = 5$.

GFather: The POS tag of the grand-father of node i. $f(t, 6) =$ *NN*.

FstRightDep: The POS tag of first dependent to the right of node i. $f(t, 6) =$ *NONE*.

Figure 3.2: Description and examples of some of the features we used. Examples are obtained using t as the tree in Figure 3.1.

Given f and V, the search space is built using a 2-step procedure. The first step defines an *initial tag* set Ci while the second uses Ci to define the family of all possible candidate tag sets in the search space. The initial tag set is built by applying $f(t, i)$ to all trees t in the tree-bank, and for their words i that have their original tags in the PTB tag set. The result of applying f to one tree t and to one word w of t whose tag belongs to V is added to Ci. Suppose that the tree in Figure 3.1 is processed for building Ci with feature *father*, and $V = \{VB\}$; then tags `VB-MD` and `VB-TO` are added to Ci. Formally, Ci is defined as $(O - V) \cup Img(f)$ where O is the set of PTB POS and $Img(f)$ is the image of f.

If original tags are replaced by the results of f, the training material can be completely retagged. In the previous example, `VB-MD` and `VB-TO` replace VB in

new POS	Feature Value	new POS	Feature Value	new POS	Feature Value
NEWTAG_1	6	NEWTAG_4	4	NEWTAG_7	11
NEWTAG_2	2	NEWTAG_5	5	NEWTAG_8	7,12
NEWTAG_3	8	NEWTAG_6	10	NEWTAG_9	17,18,20,15,24
NEWTAG_10	1,16,13,9	NEWTAG_11	0,3,14		

Table 3.2: A subset of a new POS tag set which shows entries corresponding to the new tags related to `VB` calculated with feature `Depth`.

position 2 and 6 respectively.

The second step builds the family of possible candidate tag set. The search space for feature f and tag set V is defined as all possible tags sets that are the product of merging arbitrary tags $c_1, \ldots, c_k$, c_i in Ci. Building a tag set C by merging tags $c_1, \ldots, c_k$ means that a new tag symbol t is introduced and that C is defined as $Ci - \{c_1, \ldots, c_k\} \cup \{t\}$. Merging tags also means that the training material has to be rewritten; this is done by replacing tags $c_1, \ldots c_k$ by tag t.

Since the family of possible tag sets is constructed by merging subsets of the initial tag set, the size of the search space is exponential. The traversing strategy for inspecting the search space is based on Genetic Algorithms. Genetic Algorithms need for their implementation (1) A definition of individuals: our individuals codify both a value of `alpha` to be used for building the automata and candidate tag set of the training material. (2) A fitness function defined over individuals: the quality measure q' we defined in Section 3.3. (3) A strategy for evolution: we apply two different operations to genes, namely two point crossover and mutation (see Chapter 2.2.1); crossover gets 0.95 probability of being applied, while mutation gets 0.05. We select individuals using the *roulette wheel strategy* (GC97). Finally, at each generation the population consists of 50 individuals; we let the population evolve for 100 generations.

The outcome of our optimization method is a set of new tags. The algorithm also outputs a table that assigns a new tag to each possible outcome of the feature used during the optimization procedure. This table, together with the feature f, can be used for retagging the training material with the new set of POS tags. For example, using Table A.13 together with feature `Depth` it is possible to calculate the new tag for word number 4 of Figure 3.3 (a). Note that feature `Depth`(4) returns 1 and that Table A.13 states that for words with value 1 for this feature should receive `NEWTAG_10`. Words that are originally not tagged with `VB` keep their old tags. All diferent set of POS tags induced with our algorithm are included in Appendix A.

As such, the algorithm for inducing our POS tag sets is a mixture between algorithms for inducing automata and genetic algorithms. Genetic algorithms were chosen because they provide a direct map from our problem to their representation. Genetic algorithms are used to search for the best way to merge POS tags from the initial tagging. Measure PP was chosen because it is the standard measure to evaluate automata (TDdlH00). Before trying genetic algorithms we tried out standard clustering algorithms, but they fail to capture the idea behind the q measure.

3.5 Parsing with New Sets of Tags

All tags computed in the previous section encode information regarding some syntactic feature. Even though these features are computed using dependency trees, the information codified in the new tags help phrase structure parsers to improve their performance. Our tag sets are introduced in the training material of a supervised parsing model.

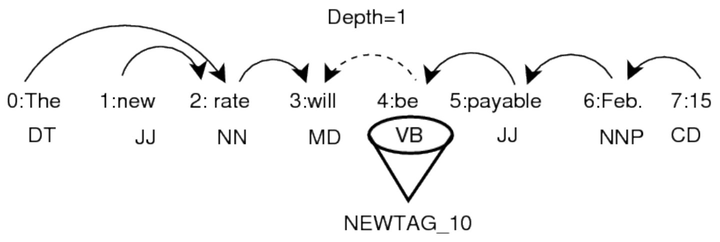
3.5.1 Rewriting the Training Material

The parser is trained in a modified version of Sections 02–21 of the PTB. All trees in those sections are modified to reflect the new set of POS tags: Our tags are introduced as new nodes above the original POS tags, as depicted in Figure 3.3 (b), by adding a new node with the new tag (in the example, `NEWTAG_10`) above the original node (`VB`). The new tag comes from the retagging schema for dependency trees returned by our optimization algorithm as explained in the previous section.

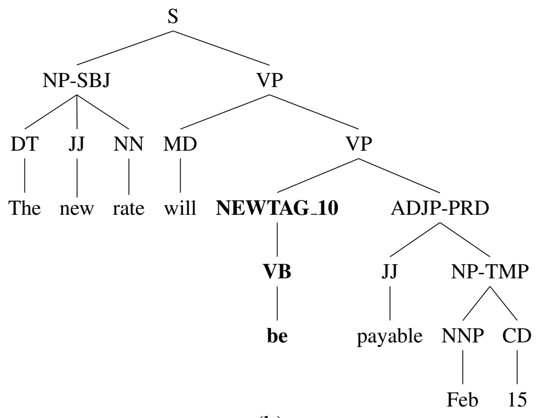
Note that instead of replacing the original POS tags with the new POS tags, we choose to add an extra-node above the original POS tag. The reason for the extra-node is given by the way the parser deals with POS tagging: The parser might guess the POS tags or they can be given with the sentence. Formally, the parser can be fed either with a sequence of pairs $\langle (w_1, c_1), \ldots, (w_k, c_k) \rangle$, w_i and c_i being words and tags respectively or with a sequence of words $\langle w_1, \ldots, w_k \rangle$.

If extra-nodes are not used in the training material, we can not resort to the first type of sequences because syntactic trees are needed to obtain the correct sequence of POS tags. In other words, in order to tag the sentence with our new POS tag set, we need to know the syntactic tree that yields the sentence, which is not available during testing. The extra-nodes are used to reflect a new set of POS tags. Adding one extra-node above each original POS tag allows reflecting one of our POS tag sets, while using two extra-nodes we can reflect two of our sets of new POS tags.

Still, without using extra-nodes, the second type of sequences can be used. We



(a)



(b)

Figure 3.3: (a) Tag `NEWTAG_10` is assigned to word number 4. (b) the new tag is introduced in the phrase structure above the original tag.

tried out replacing old tags with our new tags and we let the parser guess the correct new POS tags. Unfortunately, the performance of such approach drops dramatically. We believe that this is due to the fact that the tagger that is being used inside the parser is incapable of recovering our tags. We speculate that, since new POS tags encode syntactic information, they can not be recovered with a POS tagger; however, this point requires further research.

It is important to note that the generative model that results from training in our own version of the training material does not suffer from the extra nodes. In order to empirically try this out, we carried out a dummy experiment consisting of adding an extra node `CLUSTER` above the tag `VB` in the training material. That is, we just add an extra node that does not codify extra information. The resulting model reports the same result as the original model. This experiment shows that the generative model that is built from the material containing the extra node behaves exactly as the model without extra nodes.

3.5.2 Modifying the Parser's Training Algorithm

The parser's training algorithm was modified to make it aware of the new set of POS tags. During the training phase, Bikel's parser transforms phrase structure trees into dependency trees. During this process, the parser uses a function to determine the head symbol of each constituent. The aforementioned function uses a *head-definition table* that provides all the necessary information for finding heads in context-free grammar rules. Since entries in this table are non-terminal symbols, the table should be aware of the new set of extra-nodes that were introduced in the training material.

There are two different ways to make the head finding function aware of the new set of non-terminals: The first approach adds the new set of labels to the head-definition table. This approach is straightforward but it presents some problems when features are applied to redefine more than one syntactic category. The second approach changes the algorithm that computes heads using the head-definition table so that the new POS are completely ignored. The second approach is the one we use in our experiments.

3.5.3 Experimental Results

The results reported in this sections were performed on sentences in Section 23 of the PTB. Since the parser returns trees with extra-nodes, they are deleted *before*

Num.	Feature	Tags	L. R.	L. P.	pval R.	pval P.	F_1
	Baseline		88,53	88,63			0,8858
(1)	Depth	VB,MD	88,64	88,78	0,067	0,020	0,8871
(2)	Depth	VBN,VB,MD	88,65	88,86	0,120	0,004	0,8875
(3)	gFather	VB,MD	88,69	88,80	0,047	0,044	0,8874
(4)	gFather	VBN,VB,MD	88,64	88,80	0,160	0,047	0,8872
(5)	NumChanges	VB,MD	88,69	88,80	0,030	0,020	0,8874
(6)	NumChanges	VBN,VB,MD	88,62	88,81	0,200	0,030	0,8871
(7)	VerbAllDepth	VBN,VB,MD	88,67	88,79	0,047	0,024	0,8873
(8)	VerbDepth	VBN,VB,MD	88,67	88,78	0,069	0,051	0,8872
(9)	**VerbVBDepth**	**VBN,VB,MD**	**88,70**	**88,83**	**0,017**	**0,008**	**0,8876**
(10)	NumSib	VBN, VB,MD	88,66	88,77	0,079	0,048	0,8871
(11)	FstRightDep	VB	88,52	88,57	0,439	0,260	0,8854
(2)-(9)	Depth-VerbVBDepth	VBN,VB,MD-VBN,VB,MD	88,68	88,86	0,081	0,010	0,8876
(2)-(4)	Depth-gFather	VBN,VB,MD-VBN,VB,MD	88,62	88,82	0,210	0,020	0,8872

Table 3.3: Experiment results. The middle part shows a feature whose performance results decreases. The bottom part shows features combinations.

evaluation. Standard measures of labeled precision and labeled recall are reported.

We tested 17 different features applied to different sets of tags related to verbs. None of our experiments showed significant decrease in parsing performance, but many showed significant increases. Table 3.3 reports 11 experimental results, 10 of them show improvements. Features were selected without any optimization method. We selected features by hand using intuitions provided in (IL05) and by selecting new features that take into consideration the syntactic structure above a tree node. Such information was shown to be useful in (KM03a) and it is not being modeled by Collins' model.

Each row displays the feature that was used (see Table 3.2 for explanation), the tag set that was redefined, the results on labeled precision, labeled recall and the significance level $pval$ of the result; the latter was computed against baseline results. The baseline row reports the performance of Bikel implementation for Collins model.

The table is divided in three. The upper part shows statically significant results. The middle part provides an example of a feature that does decrease parser performance but whose decrease is not statistically significant. The bottom part reports results where features are combined with statistically significant improvements. In all cases, we consider a result statistically significant if its significance level $pval$ is below 0.05. Performance is measured using `evalb` script while significance is measured using Bikel's *Randomized Parsing Evaluation Comparator* script.

From Table 3.3, it can be seen that the combination of features (2) and (4) does not necessarily improve the results obtained by each of the features separately. For this particular example, the parser loses both performance and significance. We think that the reason for the decrease lies in the fact that the parser suffers the extra number of rules that it is has to handle. Recall from the previous section that two extra-nodes are added for every possible combination of new POS tags. Combination of features are not intended to obtain the best performance, but to investigate the impact of the combination. We tried also to combine two features by using their Cartesian product in the GA search space. In this way, we could rewrite the training material by using only one extra-node which combines both features. However, the resulting number of new tags obtained was too large and the parsing performance decreased significantly.

Our experimental results show that even though our approach hardly increases the performance figures for Bikel's parser, improvements are significant. We think that this is due to the syntactic information codified in the new set of extra-nodes. In some cases, e.g., when `gFather` feature is used, non-local information is stored in these extra-nodes. The experimental results show that not only the parser is able to recover the extra-nodes but it is also capable of taking advantage of the information stored in these nodes. Our approach minimally modifies the learning algorithm; we can state that the underlying parsing model has been left intact. Our approach mainly modifies the training material trying to incorporate information that is present there but which is not currently used by the parser.

3.6 Future Work

Our tags were tested by means of one particular parsing model. Clearly, the set of experiments we present here can be run using other parsing models. We think that such experiments can help understanding how different models take advantage of the information that is coded in our non-terminals. Since our tags are built using dependency trees, we believe that they can better help parsers that do not rely so heavily on dependencies e.g. (Cha00, HT05) as Collins model. This is because as they do not use dependency structures, our POS tags may help to add the syntactical information obtained from dependencies. During our experiments we could not directly replace old POS tags with our new tags because the parser's built in tagger did not handle our tags correctly. This puts forward questions that require further research, such as: can we build POS taggers that are able to recover complex tags that encode deep syntactic information? or, more generally, which information can be coded in tags so that it can be recovered using taggers? Some answers to this question have been given in the literature (Osb00).

Another experiment that could be interesting to perform consists in using the MLE estimator as the function to be optimized, instead of the perplexity of the automaton; or in using a combination of both. The reason is that, although calculating perplexity is the standard procedure to qualify automata, the MLE estimator is considered standard in syntactic parsing for an unsupervised measure of the quality of a treebank. The calculation of MLE can be performed computing the average of probabilities of the "dependency words" included in the training material, which is also used to calculate the Missed Samples quality measure. Finally, note that measures used in this work are principally related to "how well the automaton reflex the training material". However, a very interesting measure with a more general purpose such as Minimum Description Length (MDL) (Ris78)[1], could help us to determine "how good is the automaton itself". This notion could be a third measure to be combined.

3.7 Discussion and Conclusions

When we have to decide which tag set is the best for our task, we should take into account the purpose of the tags sets and the results obtained from an evaluation methodology which gives us an idea of how good these tag sets are for that specific task. As an example, the purpose of LOB corpus (Mar83) was to be used in English language teaching and research, thus the designers developed a comparatively complex tag set to reflect fine distinctions of English grammar for learners and teachers. In the case of PTB corpus, tags were mainly intended to be used in language engineering; for example, as a training set for Machine Learning systems because they would work better with smaller tag sets. In this work we adopt a similar criterion for evaluating POS tag sets based on how much they improve the performance of syntactic parsers, coupled with the test sample perplexity.

In general the POS sets are far too general to capture the syntactic information of words. Taken to an extreme, we can say that each word may have its own unique syntactic information, especially for modern parsers where a lexicalized grammar is used. However, in practice, given the limited training data available and the resulting data sparseness, the syntactic relationship between words may be more

[1]MDL is a formalization in which the best hypothesis for a given set of data is the one that minimizes the amount of information for both data and hypothesis.

accurately modeled at a partition level between POS tags and the lexical identity of each word. In this work, we aim to find the more accurate POS tag set for syntactic parsers, by implementing an optimization procedure which splits and merges tags according to the similarity of the syntactic behavior of words. These behaviors were characterized by means of probabilistic regular automata in a dependency syntax setting.

At this stage of the NLP research area, it is known that a naive PCFG which induces its rules in a canonical way does not perform well. This naive grammar yields a low performance because, as it is well summarized in (PBK06), its context-freeness assumptions are too strong in some places (e.g. it assumes that subject and object NPs share the same distribution) and too weak in others (e.g. it assumes that long rewrites are not decomposable into smaller steps). As a consequence, better results are obtained by passing contextual clues through tag labels, and this gain in performance is bounded by the problem of sparse statistics. We propose in this work, to add structural information to the syntactic categories tag set, and in this way, feed the parser with that information. Our results show that the resulting tags encode structural syntactic information that was used by an state-of-the-art parser to significantly improve its performance.

Chapter 4

Looking for the best language for dependency structure

Head finders have been usually inspired on linguistic bases and they have been used by parsers as such. In this Chapter, we present an optimization set-up that tries to produce a head finder algorithm that is optimal for syntactic parsing. This idea arises from the work described in the previous Chapter, as we found that improvement in parsing performance can be achieved by changing the head finder to make it aware of the new POS tags. We then sought to appraise the impact of head finders in parsing and whether better head finding rules could be automatically derived. All the results presented in this Chapter were published in (DIL10).

4.1 Introduction

Head-finder algorithms are used by supervised syntactic parsers to transform phrase structure trees into dependency ones. The transformation is carried out by selecting a word as the head in every constituent. Head-finder algorithms are based on a set of head-finder rules which provides instructions on how to find the head for every type of constituent. For every internal node of a tree, the head-finder rules specify which children of the node contains the head word. The first set of head-rules, based on linguistic principles, was introduced in (Mag95a) and it is used by many state-of-the-art statistical parsers, like (Cha00, KM03a, Bik04b, Col97) with only minimal changes.

The standard set of head-finder rules was handcrafted and, consequently, not optimized for parsing; therefore, there might exist different sets of head-finder rules that can improve parsing performance. In this chapter we investigate their role in parsing and we experiment with two different state of the art parsers. We present an optimization algorithm that improves the standard set of head finders, one rule at the time, with the goal of finding an optimal set of rules. Even though our optimization algorithm produces statistically significant improvements, it hardly obtains a better performance. In order to better understand why our optimization algorithm cannot produce bigger improvements, we test the stability of the search space. We test this by generating different head finders: we generate head finders that always select the rightmost and leftmost sub-trees as the trees containing the headword. We also generate 137 random sets of rules, and we test head finders that are not consistent, that is, head finders whose set of rules changes during the same training session.

Our optimization procedure aims at finding the best possible set of rules that improves parsing performance. Our procedure is defined as an optimization problem, and as such, it defines the quality measure that it has to optimize, its search space, and the strategy it should follow to find the optimal set of rules among all possible solutions. The *search space* is the possible sets of rules; our procedure optimizes one rule in the set at a time. A new set of rules is then created by replacing an original rule in the standard set with its optimized rule. The *quality measure* for a rule set is computed in a serie of steps. First, the training material is transformed from phrase structure trees into dependency ones using the rule to be evaluated; second, a bilexical grammar (Eis97) is induced from the dependency tree bank, and finally, the quality of the bilexical grammar is evaluated. The quality of the grammar is given by the perplexity (PP) and missed samples (MS) found in the automata of the grammar as explained in Section 4.4. Finally, the *strategy* for traversing the search space is implemented by means of Genetic Algorithms. Once we obtain an optimized set of rules, we proceed to evaluate its impact in two parsers, Collins' parser (Col97) by means of Bikel's implementation (Bik04b), and the Stanford parser (KM01).

The source code of these two parsers is available and their head finder algorithms are rather easy to modify. We considered experimenting also with the Maltparser (NHN$^+$07) but its performance is hard to evaluate when its head finder is modified. Our experiments show that the parsing performance of the two parsers is insensitive to variations in head finders. They also show that among all possible head-finders, our optimization procedure is capable of finding improvements. Our experiments also show that in the presence of inconsistent head finder rules, parsers performance drops 1.6% and 0.9% for Bikel's and for Stanford respectively. Our experimental results with random head finders show that modifications in the rule for `VP` produced the biggest impact in the performance of the two parsers. More interestingly, our experiments show that inconsistent head finders are more stable than random deterministic head finder. We argue that this is the case because the variance on the structures the later produces is considerably bigger with respect to the former. Our experiments also show that Stanford parser performance is more stable with respect to variations in the head finder rules than Bikel's.

All in all, our experiments show that, even though it is possible to find some new set of rules that improves parsers performance, head finding algorithms do not have a decisive impact on the performance of these two state-of-the-art syntactic parsers; this also indicates that the reason for their performance lies beyond the procedure that is used to obtain dependencies.

The rest of the Chapter is organized as follows. Section 4.3 explains head finding algorithms. Section 4.4 presents the quality measure used in our optimization algorithm, while Section 4.5 discusses the search space and the strategy to traverse it. Section 4.6 presents how random rules are generated, Section 4.7 presents the results of our experiments and Section 4.2 introduces related work. Finally, in Section 4.8 we include a discussion about the contribution of our work, and also, concludes the Chapter.

4.2 Related Work

Similar work has been published in (CB02), and an improved version can be found in the Bikel's thesis (Bik04b). In this work, the authors tried to induce head rules by means of defining a generative model that starts with an initial set of rules and uses an EM-like algorithm to produce a new set of rules that maximize the likelihood. They used different sets of rules as seeds for the EM, but the approach only shows improvement when the standard set of rules is used. In contrast to our approach, none of their improvements were statistically significant. They also show that when the seed is a set of random rules, the overall performance decreases.

In a different approach (SZ09) the authors present different unsupervised algorithms for head assignments used in their Lexicalized Tree Substitution Grammars. They study different types of algorithms based on entropy minimization, familiarity maximization, and several variants of these algorithms. Their results show that using the head finder they induced, an improvement of 4% over a PCFG parser using standard head assignments is obteined. In our work, we don't use lexicalized grammars. Our approach is based on improvements to a given rule set, as opposed to theirs where they use unsupervised methods to find assignments for heads.

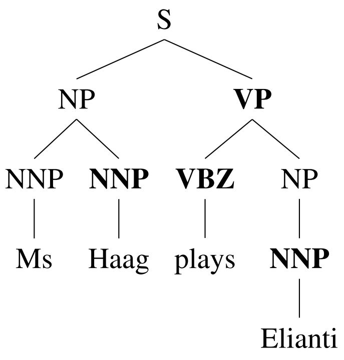


Figure 4.1: Sentence 2 from Section 2 of the PTB. The nodes where the head of each constituent is searched for are marked in boldface.

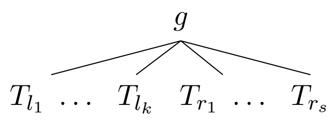


Figure 4.2: A simple phrase structure tree.

4.3 Head Finding Algorithms

For each internal node of a phrase structure tree, the head finder (HF), determines which of its subtrees contains the head word of the constituent. The procedure of transforming a phrase structure into a dependency one starts in the root of the tree and moves downwards up to the tree preterminals. The HF has as a parameter a set of head finder rules R. R contains one rule for each possible grammatical category. Formally, let R be $\{r_{gc_1},\ ...,\ r_{gc_k}\}$, where r_{gc_i} is the head-finder rule associated with the grammatical category gc_i; the set $\{gc_1,\ ...,\ gc_k\}$ is the set of all grammatical category tags like `S, VP, ADJP, ADVP, SBAR,` for the Penn Tree-Bank (MS93) (PTB).

A head finding rule r_{gc} is a vector of pairs $((d_1, t_1), \ldots, (d_{k_n}, t_{k_n}))$, where t_i is a non-terminal and $d_m \in \{left,\ right\}$ is a *search direction.* We also use the notation of *direction vector* to refer to the vector $(d_1, \ldots, d_{k_i})$ which is the projection of the first component of the head-finding rule vector. Similarly, the *tags vector* $(t_1, \ldots, t_{k_i})$ is the projection of the second component. We refer to a head-finder rule as one vector of pairs, or as a pair of vectors. For example, in Figure 4.3 is shown the rule that is associated with tag `VP` in the standard set of head-finder rules. Note that, in the standar set of rules `l` is used to represent the *left* direction, and similarly `r` is used for the *right* one.

It is important to highlight that our definition of head-finder rule is a simplification of the standard head-finder rule. In the standard definition of rules for tags `NP`

```
((l TO)(l VBD)(l VBN)(l MD)(l VBZ)(l VB)(l VBG)(l VBP)(l VP)(l ADJP)(l NN)(l
NNS)(l NP))
```

Figure 4.3: An example of Head Rule.

and NX, sets of non-terminals are used instead of simple non-terminals. Our definition excludes this situation because, otherwise, the size of the search space makes the optimization procedure unfeasible.
In order to show how the head finder algorithm works, we introduce a few auxiliary functions. $root(T)$ returns the root node of the phrase structure tree T; $children(T, g)$ returns the list of children of node g in T and, $subtreeList(T)$ returns the list of sub-trees of T ordered from left to right. For example, if T is the tree in Figure 4.2 then $root(T) = g$,
$children(T, g) = [root(T_{i_1}), \dots, root(T_{l_k}), root(T_{r_1}), \dots, root(T_{r_s})]$ and $subtreeList(T) = [T_{l_1}, \dots, T_{l_k}, T_{r_1}, \dots, T_{r_s}]$. Using this definition, we formally define in Algorithm 6 the algorithm HF_R that transforms a phrase structure tree into a dependency one, where R is a set of head finder rules.

Consider the tree in Figure 4.1. Suppose that the head-finder rule for tag VP is ((l TO)(l VBD)(l VBN)(l MD)(l VBZ)(l VB)(l VBG)(l VBP)(l VP)(l ADJP)(l NN)(l NNS)(l NP)). When the head finder algorithm reaches the node VP, it looks from left to right a tag TO; since it cannot find such tag, it looks from left to right a tag VBD, it keeps changing what it is looking for until it looks from left to right for a tag VBZ. Once it has found it, it marks that subtree as head and it recursively inspects all subtrees.

4.4 A quality measure for Head Finder Rules

This section introduces the quality measure used in our optimization procedure for finding the best set of head finder rules for syntactic parsers. Our procedure is based on the optimization of a quality measure q defined over a set of head-finder rules.

The measure q of a set of head-finder rules is defined as a measure of the grammar that is built from using the head-finder rule to transform the PTB into dependencies. The measure is then defined over the automata that defined the grammars. The measure over bilexical grammars contains two components: *test sample perplexity* (PP) and *missed samples* (MS), with the same idea that in Section 3.3.2.

```
Data: the input tree $T$, $r_{gc_i} = ((d_1, t_1), .., (d_{k_i}, t_{k_i}))$ is a rule from $R$
      defined for tag $root(T) = g_r$
Result: the tree $T$ with the head $marked$
1  $s = length(children(T, g_r))$;
2  foreach $(d_l, t_l) \in r_{gc_i}$ do
       /* iterate in each element of the rule */
3      if $d_i$ == left then
4          for $j = 1$ to $s$; $j = j + 1$ do
               /* seek from left to right in $children(T, g_{root})$ */
5              if $children(T, g_{root})[j]$ == $t_i$ then
                   /* if the tag matches, the head is marked. */
6                  $Mark(children(T, g_{root})[j])$;
7              end
8          end
9      end
10     else
11         for $j = s$ to $1$; $j = j - 1$ do
               /* seek from right to left in $children(T, g_{root})$ */
12             if $children(T, g_{root})[j]$ == $t_i$ then
                   /* if the tag matches, the head is marked. */
13                 $Mark(children(T, g_{root})[j])$;
14             end
15         end
16     end
17     ;
18 end
19 foreach $T_k \in subtreeList(T)$ do
       /* recursively call to subtrees */
20     $HF_R(T_k)$;
21 end
```

Algorithm 6: The Head Finder Algorithm.

Better values of $\mathrm{MS}$and $\mathrm{PP}$for a grammar mean that its automata capture better the regular language of dependents by producing most strings in the automata target languages with fewer levels of perplexity. The quality measure of grammar is then the mean of PP's and MS's for all automata in the grammar.

In order to compute q for a given set of rules, we proceed in a similar way that of the previous Chapter. We transform Sections 01-22 of PTB into dependency structures. Using the resulting dependency treebank we build a bilexical grammar, and finally, we compute a quality measure on this grammar. The test sample that is used to compute PP and MS comes from all trees in sections 00-01 of the PTB. These trees are transformed to dependency ones by using HF_{R_c}, where R_c is the candidate set of rules.

4.5 A Genetic Algorithm Set-Up for Head Rules

This section introduces the search space and the strategy to traverse it in our optimization procedure. The search space consists of different sets of head rules. The standard set of rules contains 26 rules. The longest is the one associated with `ADJP`; it contains 18 entries. Finding a new set of rules means that we should find a new set of 26 vectors. For each candidate rule we have to transform the PTB, build the bilexical grammar, and compute $\mathrm{PP}$and $\mathrm{MS}$for all automata. Our algorithm takes 1.2 minutes to evaluate one candidate set of rules.

In principle, all possible head-rules can be candidate rules, but then the search space would be huge and it would be computationally unfeasible to traverse it. In order to avoid such search space, we run a series of experiments where we optimize one rule at a time. For example, one of our experiments is to optimize the rule associated with `VB`. Our search space contains all possible set of rules where all rules except the one associated to `VB` are as in the standard set of head-finder rules.

To optimize one rule we traverse the search space with Genetic Algorithms. Genetic Algorithms need for their implementation (1) Definition of individuals: each individual codifies one candidate of the head-finder rule that is being optimize. (2) A fitness function defined over individuals: the quality measure is computed by constructing a set of head-finder rules by adding the candidate rule to the standard set of rules, building a bilexical grammatical and evaluating it as described in Section 4.4. Finally, (3) A strategy for evolution: we apply two different operations to individuals: ad-hoc crossover and mutation; crossover gets 0.95 probability of being applied, while mutation gets 0.05. We test two different selection strategies: *roulette wheel strategy* and *tournement selection* (see Chapter 2.2.1), and we ob-

Tag array:

TO	VBD	VBN	MD	VBZ	VB	VBG	VBP	VP	ADJP	NN	NNS	NP
0	1	2	3	4	5	6	7	8	9	10	11	12

Direction array:

false	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*
0	1	2	3	4	5	6	7	8	9	10	11	12

Figure 4.4: The algorithmic representation of an individual corresponding to the Head Rule from Figure 4.3.

tained similar results. In our experiments, in each generation there is a population of 50 individuals; we let the population evolve for 100 generations.

Our definition of individuals codifies a head rule. We define an individual as two arrays, one of type $String$[1] which represents the *tag vector* of a rule, and a $boolean$ array that represents its *direction vector*. Similarly we map the direction vector in the $boolean$ array and, $true$ maps with $left$ and $false$ with $right$. In Figure 4.4 we show how a head rule from Figure 4.3 is represented as an individual of the GA. Note that if we permute elements from arrays, this means permutations in the rule definition.

The *mutation* function is easily defined by computing a random permutation of the rule tags vector and a random sample of its direction vector. The *crossover* operation is defined as follows. Let $[g_1, \dots, g_i, \dots, g_n]$ and $[h_1, \dots, h_i, \dots, h_n]$ be the tag vectors of two different individuals and let i be random number between 1 and n. The crossover produces two new individuals. The tag vector of one of the individuals is defined as follows:

$$sub([g_1, \dots, g_n], [h_{i+1}, \dots, h_n]) \cdot [h_{i+1}, \dots, h_n]$$

where the operator $(\cdot)$ appends two arrays, and $sub(x, y)$ deletes the elements in x that are in y. The tag vector of the other individual is defined similarly by changing g by h and vice versa. The *direction vectors* of the new individuals are obtained by using the usual definition of single point crossover for boolean vector. In this way,

[1]In fact, the implementation uses $Integer$, and there is a mapping function which assigns each number from the array vector with the coordinate of the corresponding tag in the original *tag vector*. We use $String$ to make it easy for explanation, and it is just an implementation detail.

crossover ensures that the resulting head rules do not have repeated tags.

4.6 Stability of Head Finders

As it is shown in Section 4.7, our optimization method only improves the performance of Bikel's parser. The reason for the lack of improvement of Stanford parser can be either because our optimization method is ill defined or because the parser is indifferent to the set of head-finders rules. However, using no head finder, that is, non dependency grammars, performance never reaches beyond 75%. So, heads and dependencies based on heads are an important element in parsing performance.

In this section we present experiments that try to shed light on this issue. We tested (1) randomly generated head finder rules, (2) head finders whose rules were reversed, (3) head finders that always choose right or left, and (4) inconsistent head finders.

Random Head Finders: We experiment generating several sets of head finder rules. A random set of head finder rules is created by replacing one rule in the standard set of head finder rules by one random rule. A random rule is created by randomly permuting the elements of both the tags vector and the direction vector. Experimental results for this head finder are shown in Table 4.1 (A).

In Figure 4.5, the first row shows the head rule defined in the standard set for category S. The second row shows a random permutation of this rule. The last row shows a reverse permutation of the original one. The reverse permutation of a rule is obtained by reversing the order of its *tag vector* and leaving its *direction vector* unchanged. In this example, the rule presented in the second row is calculated using the permutation $[7, 1, 4, 8, 2, 3, 6, 5]$ for the *tags vector* and its random *direction vector* was (l, r, r, l, l, l, l, l). We generate 119 rules and, consequently, 119 different head finder rule sets. Each of these sets differs in one rule from the standard set. In this way, we show the impact of each rule in the overall parsing performance.

We also experiment with a set of rules were *all* of its rules were randomly generated. We test 7 of such random sets for each parser. Experimental results for this head finder are shown in Table 4.3.

Reverse Rules: There are 26 rules in the standard set of head finder rules. We generate 26 new sets by changing one rule at a time by its reverse. The reverse of

Original	`(S (l TO)(l IN)(l VP)(l S)(l SBAR)(l ADJP) (l UCP)(l NP))`
sampled rule	`(S (l UCP)(r TO)(r S)(l NP)(l IN)(l VP) (l ADJP)(l SBAR))`
reverse rule	`(S (l NP)(l UCP)(l ADJP)(l SBAR)(l S)(l VP) (l IN)(l TO))`

Figure 4.5: The first row shows the original Collins's head rule for `S`. The second row shows a random permutation of the original rule. The last row is the reverse of the original rule.

a rule is constructed by reading it from left to right. The impact of reverse rules in parsing performance is shown in Table 4.1 (B).

Leftmost and Rightmost Head Finders: We define two special algorithms for finding heads. The always-leftmost and always-rightmost algorithm chooses for each internal node the leftmost and rightmost subtree respectively. These are special cases of head finder algorithms that cannot be expressed with a set of rules. In order to implement these algorithms, we modified both parser implementations. The results are shown in Table 4.2 (B).

Non-deterministic Head Finders: All previous experiments were based on deterministic head finders: every time they are used to transform a given phrase structure tree, they transform into the same dependency tree. We implemented a non-deterministic head finder algorithm, this algorithm flips a coin every time it has to decided where the head is. When this head finder is used to transform a phrase structure into a dependency tree it produces different dependency trees for every time it is called. We report results for 7 of these experiments for each parser, they can be seen in Table 4.3.

4.7 Experimental Results

In this section, we analyze the results of all our experiments. In all experiments we used Sections 02-21 of the PTB for training and Section 23 for testing. The optimization algorithm use Sections 00-01 for computing the quality measure defined on automata. Our experiments aim to analyze the variation in performance by

	Bikel			Stanford		
rule tag	**avg.**	**max.**	**min.**	**avg.**	**max.**	**min.**
WHADJP	88,589	88,596	88,583	85,739	85,739	85,739
CONJP	88,586	88,601	88,582	85,739	85,739	85,739
WHNP	88,586	88,596	88,577	85,739	85,739	85,739
SINV	88,485	88,608	88,009	85,749	85,772	85,730
QP	88,538	88,604	88,474	85,747	85,759	85,732
RRC	88,588	88,595	88,583	85,739	85,739	85,739
S	88,092	88,569	87,458	85,689	85,726	85,652
ADVP	88,586	88,615	88,564	85,740	85,743	85,739
NAC	88,594	88,607	88,581	85,743	85,743	85,743
SBAR	88,195	88,653	88,013	85,734	85,739	85,733
VP	**87,330**	**88,471**	**85,870**	**85,247**	**85,612**	**84,918**
SQ	88,571	88,592	88,562	85,739	85,739	85,739
ADJP	88,616	88,698	88,566	85,727	85,739	85,718
WHPP	88,583	88,583	88,583	85,739	85,739	85,739
SBARQ	88,583	88,583	88,583	85,739	85,739	85,739
PP	88,617	88,668	88,583	85,740	85,740	85,739
WHADVP	88,607	88,706	88,583	85,739	85,739	85,739

(A)

Gram.tag	F_1^B	F_1^S
WHADJP	88,596	85,739
SBAR	87,990	85,733
CONJP	88,600	85,739
VP	85,820	84,249
WHNP	88,596	85,739
SQ	88,562	85,739
SINV	88,465	85,758
ADJP	88,626	85,726
QP	88,490	85,748
WHPP	88.583	85,739
RRC	88,595	85,739
SBARQ	88.583	85,739
S	88,178	85,670
PP	88.583	85,739
ADVP	88,613	85,739
WHADVP	88.593	85,739
NAC	88,603	85,743

(B)

Table 4.1: (A) Parsing results obtained by replacing one rule in the standard set by a random rule. Each row shows the average, maximal and minimal impact in the F_1 measure for each parser. (B) Experiments result for each Head finder built with the reverse of head rule. One column for each parser.

changing one or more head rules in the standard set of head rules. The rules that are modified by our experiments correspond to tags:
{`WHADJP, CONJP, WHNP, SINV, QP, RRC, S, ADVP, NAC, SBAR, VP, SQ, ADJP, WHPP, SBARQ, PP, WHADVP`}.

Table 4.1 (A) include the parsing results obtained by replacing one rule in the standard set by a random rule. Each row of this table shows the average, maximal and minimal impact in the F_1 measure for the considerd parsers. Table 4.1 (B) shows F_1 measure for the 17 rules that were obtained by reversing one of the rules in the standard set at a time.

From Tables 4.1 (A) and (B) we can see that the rule defined for tag `VP` has the greatest impact on the performance of both parsers.

Table 4.2 (A) shows the performance of the set of rules produced by the optimization procedure. Each row displays labeled precision, labeled recall, significance level $pval$, and harmonic mean F_1. The baseline row reports the performance of both parsers using the standard set of rules. $pval$ was computed against the baseline. We consider a result as statistically significant if its significance level $pval$ is below 0.05. Performance value were computed using the `evalb` script,

Num.	**L. R.**	**L. P.**	**pval R.**	**pval P.**	**F_1**
Bikel Baseline	88,53	88,63			88,583
optimal head	88,72	88,85	0,006	0,002	88,785
Stanford Baseline	85,26	86,23			85,742
optimal head	85,24	86,22	0,098	0,131	85,727

(A)

Parser	F_1 worst choice	F_1 Rightmost	F_1 Leftmost
Bikel	82,486	83,024	85,102
Stanford	84,092	84,206	85,566

(B)

Table 4.2: **(A)** The result of the experiments corresponding to the optimized head finder. The upper part shows evaluation in Bikel's parser, while the bottom with Stanford parser.**(B)** First column shows the F_1 when all worst performing rules, reported in Table 4.1 (A), are put together. Second and third columns show average F_1 for the always right-most, and always left most head finders.

significance values were measured using Bikel's *Randomized Parsing Evaluation Comparator* script. The table shows that the performance in the Stanford parser using our optimized head finder set of rules is below the baseline, however, this decrease in performance is not statistically significant. The best set of head rules was obtained by combining all rules that our optimization method produced. Table 4.3 shows the results of the random head rule generation. The Table contains one row per each rule that was permuted. We consider 17 different rules, for each, we build $7*17$ new random rules. Each row shows the maximal, the minimal and the average F_1 measure we obtained.

The first column of Table 4.2 (B) shows the results for the head finder that is built by using the 17 rules with the worst performance in experiments in Table 4.3. The second and the third columns show the results for the head finder algorithms that choose always the leftmost and always the rightmost respectively.

Table 4.3 shows results for the non-deterministic and deterministic head finders. The set of rules are obtained by changing rules for the 17 tags considered in our work. We run 7 tests for deterministic and 7 tests for non-deterministic head finders. In both cases, we calculate the average, the maximum and the minimum, obtained for the measure F_1. The results show that the non-deterministic head finder

	Rand. No Det.			Random Det		
Parser	**avg.**	**max.**	**min.**	**avg.**	**max.**	**min.**
Bikel	86,976	87,166	86,754	86,001	87,974	83,857
Stanford	84,810	84,997	84,691	84,805	85,625	84,360

Table 4.3: Experiments result of random choice of rules, for each experiment we show the impact in the F_1 measure for the average, maximal and minimal.

is more stable because the variation between the minimum and the maximum results is lower. A priori, this is a surprising result, because the dependency trees used to induce the grammar during the training phase have percolated inconsistent heads. We think that the non-deterministic head finders are more stable because, in average, they make more "correct" choices. In contrast, if deterministic head finders contain an erroneous rule, all the resulting dependency trees are wrong. This fact is also supported by the results reported in Table 4.2 (B). It shows that using the head finder built out of the worst performing rules is the one with the worst performance in both parsers. The performance drops nearly 6% and 1.9% for Bikel and Stanford respectively. The right-most head finder decline is the next considering the performance downfall.

4.8 Discussion and Conclusions

Algorithms for automatically choosing heads are very important both for NLP, and for linguistic theory. In our approach, we try to find the "optimal" way of choosing heads for syntactic parsers. To do so, we vary the head rules that are used by the head finders algorithms when transforming constituent trees into dependency ones.

The evaluation of the assignments of heads in each constituent is not easy, especially because there is not an agreement in what a "gold standard" (SZ09) should be. For example, to evaluate a dependency annotated version of the PTB, we first transform it from its original constituent form using classical Magerman-Collins rules. In contrast, the Tiger corpus can be compared directly since it is already annotated with dependencies. Moreover, the criteria for the dependency annotation also varies in the two methods mentioned (SZ09). Instead of these types of evaluation, we choose to evaluate the head assignment indirectly, by evaluating the labeled precision and labeled recall reached by the parser when our optimized rules are used. This criteria of evaluation is taken, because making an evaluation using the rules we are trying to optimize would have no sense.

It is important to mention that in (Bik04c) the author does not perform a complete analysis on how each head rule affects the parsing performance, although he studies how the choice of heads affects the Collins model. Moreover, he bases his studies on just one parser. In our work, we decide to make a more in depth analysis on how the heads selection helps the syntactic parsing performance: on the one hand, we try to find which is the maximal performance reached by the parser changing the heads rules. On the other hand, we aim to quantify how the choice of heads affects the parsing performance, not only for Collins' but also for Stanford parser.

We find some new rules which hardly improve parsers performance. Just like the previously mentioned researchers, we find that variations in head finding algorithms do not have a decisive impact on the performance of syntactic parsers. The variations have less impact than expected, to the extent that, as long as it preserves constituency, an aleatory translation of the phrase structure trees into dependency ones can be used without damaging considerably the parsing performance. We also make a complete analysis on how each head rule defined for each grammatical category affects the parsing performance. We find that the choice of heads in a verb phrase, even without looking further at NN tags, have the more relevant impact.

The sensibility to the change of head rules among the diferent parsers studied is somewhat different. The main finding is that Collins' parser is much more sensitive to the change of rules. For example, the variance between the worst and best performance is about 6.5%, while in Stanford parser, the variance is only around 2%. We think this difference lies in that Collins' model is fully lexicalized which makes it much more sensitive to the changes of the heads, due to data sparseness.

Finally, removing head finding altogether produces about a 12% decrease in performance, considerably higher than the 1.9% and 6% decreases in performance produced by the worst possible head finders. Therefore, as in (Bik04c) we reach at similar conclusions, that is, head finders are crucial for the performance of dependency parsers but their variations are not. We think that the use of dependency structure is crucial in syntactic parsing because it encodes contextual information, which standard PCFGs do not have. In addition, another reason to take into account is that as the measures are calculated over constituent trees and many dependency trees may have the same corresponding constituent one, different ways of choosing heads may produce the same results.

Chapter 5

Optimizing Automata for Unsupervised Dependency Parsing

Unsupervised parsing induction has attracted a significant amount of attention over the last few years. However, current systems exhibit a degree of complexity that can shy away newcomers to the field. In this Chapter we present a new unsupevised parser which is fully fleshed and easily reproducible. We experiment in eight languages that inform intuitions in training-size dependent parameterization. All the results included in this Chapter where published in (DIL11, DILL10).

5.1 Introduction

Over the last years, unsupervised dependency parsing has acquired increasing relevance (KM04, SE05, CGS08, CS09, HJM09, SAJ10a, GGG^{+}10). Unsupervised methods, in contrast with supervised and semi-supervised methods, do not require training from hand-annotated corpora which is usually expensive. Therefore, unsupervised parsing might be a solution for languages and domains with minimal resources.

Currently dependency parsers exhibit a degree of complexity that can shy away newcomers to the field. We challenge such complexity and present a straightforward weak-EM based system. We achieve results close to state-of-the-art ones, while making it simple to experiment with sub-components (see below).

In unsupervised dependency parsing the goal is to produce dependency relations such as those shown in Figure 5.1, where each arc is a relation between a head and its argument. Since the task is unsupervised, correct dependency structures are

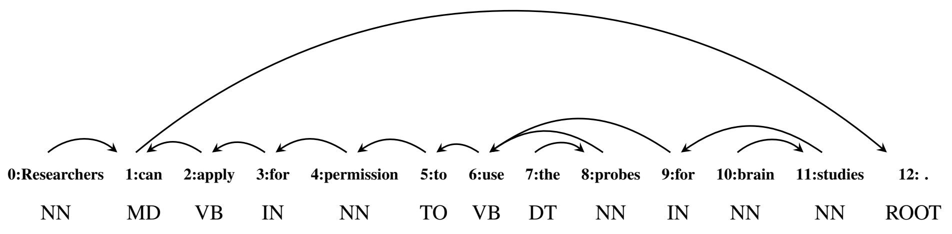


Figure 5.1: Tree extracted from the PTB, file `wsj_0297.mrg` and transformed to a dependency tree.

not available and our input consists only of sequences of parts of speech (POS) tags. Our dependency relations are modeled with Probabilistic Bilexical Grammars (PBGs) (Eis96) for which we have implemented a novel learning algorithm. Our algorithm is a weak version of the EM-algorithm (DLR77). As shown in (Pre01) an EM algorithm for inducing grammars can be described as an iteration between an E-step and an M-step. During the E-step a new treebank is computed, while during M-step a grammar together with its probabilities is read out from the treebank. Usual implementations of the EM do not actually compute the treebank but they compute the new grammar using inside-outside probabilities from the previous step.

We take a different approach. We present an algorithm based on 4 different modules that mainly computes an approximation of the treebank. These 4 components are: a supervised PBGs INDUCTOR (simulating the M-step), a k-BEST PBG PARSER, plus a TREEBANK REPLICATOR (together simulating the E-step), and an initial TREEBANK GENERATOR (in charge of building the initial seed). The resulting weak-EM algorithm is well defined for different PBG learning algorithms and for different initial treebanks. Consequently, the componets PBGs INDUCTOR and TREEBANK GENERATOR can be instantiated differently at almost no effort.

Thanks to the versatility offered by our schema, we are able to test three different ways to generate initial treebanks, and two different schemas for learning automata. Most of the recent work in this area, e.g., (HJM09, CS09, SAJ10a), has focused on variants of the Dependency Model with Valence (DMV) (Kle05). DMV was the first unsupervised dependency grammar induction algorithm to achieve accuracy above a right-branching baseline. With all its strengths, DMV is still limited in the type of dependencies it can model. The DMV model can be seen as a sort of PBG with the particularity that all of its automata have similar structures and that they only differ in the probabilities of their arcs. In contrast with our model, DMV and others in the literature are still in need of a well understood learning mechanism.

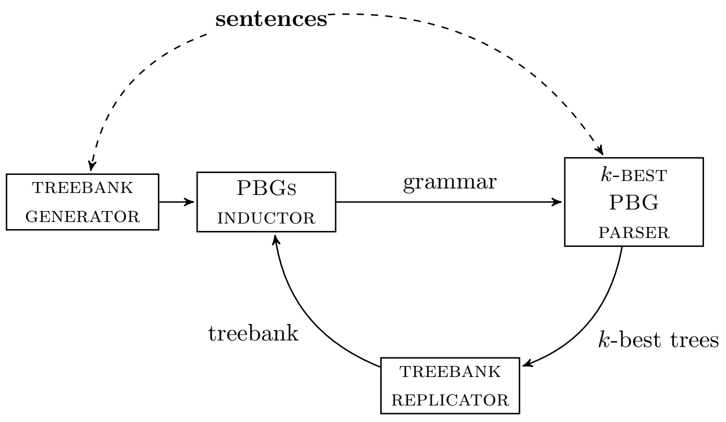


Figure 5.2: The general weak-EM algorithm.

By using a generalization of EM we can tap into a large body of learning expertise.

Our results show a very good performance in 5 languages. Particularly, for English these are very close to the state-of-the-art performance for sentences with a restricted length of up to 10 POS. For languages with enough available training material (German, Portuguese and Danish), we have state-of-the-art results or close to them such as for Swedish. For the rest of languages, our performance is considerably higher than the standard DMV performance.

This Chapter is organized as follows: Sections 5.2 and 5.3 present our framework and the algorithms for learning automata. Section 5.4 shows experimental results, Section 5.5 discuses related work, 5.6 describes some research lines which may improve our parser. Finally, Section 5.7 discusses and concludes the Chapter.

5.2 Learning Architecture

The learning algorithm (Figure 5.2) consists of 4 modules: the TREEBANK GENERATOR, the PBGs INDUCTOR, a k-BEST PBG PARSER, and a TREEBANK REPLICATOR. The learning algorithm starts by creating a treebank over a given set of sentences. The resulting treebank is used by the PBGs INDUCTOR module to induce a PBG. Once a grammar has been induced, it is used by the k-BEST PBG PARSER to parse all original sentences. The k-BEST PBG PARSER returns the k-best trees for each sentence with their corresponding probabilities. All these trees are used by the TREE-

BANK GENERATOR to create a new treebank that reflects the probabilities of all trees. Once the new treebank has been created, the algorithm cycles between the PBGs INDUCTOR, k-BEST PBG PARSER and TREEBANK REPLICATOR until the likelihood of the k-best trees hopefully converges. We will now describe each component.

PBGs INDUCTOR. This module is one of the key components of our algorithm. Its task is to find a probabilistic bilexical grammar from a collection of dependency trees. We use the same ideas explained in Section 3.3.1 to induce a PBG $B = (W, \{r_w\}_{w \in W}, \{l_w\}_{w \in W},)$, but we try with different mechanism to learn automata. In Section 5.3, we describe some algorithms capable of induce the automata l_w and r_w.

k-BEST PBG PARSER. Since PBGs are a particular case of probabilistic context free grammars, our parser for PBG is based on an implementation of a k-best CYK parser for Chomsky Normal Form PCFGs. The k-BEST PBG PARSER returns the k-best trees together with their probabilities.

TREEBANK REPLICATOR. Intuitively, this module uses the k-best trees for creating a new treebank that resembles the known probabilities of individual trees. Although we know the probabilities of the sentences, we need to replicate them because it allows us to use any automata inductor, for example MDI, which accepts only a set of sentences as a training material. Since the grammar inductor only takes a treebank as input, it is not aware of their probabilities. The TREEBANK REPLICATOR module replicates the k-best trees in such a way that the probability mass associated to each tree is proportional to the probability assigned by the parser. The TREEBANK REPLICATOR produces a new treebank that contains as many trees for each sentence as are required to reproduce the sentence probability. In order to mark the boundaries of the number of possible replicas, we introduce a constant M that states the maximum number of trees a sentence will have in the resulting treebank. Suppose that $C = \{c^1 \ldots c^N\}$ of N sentences are the input senteces. Suppose also that $t_1^j \ldots t_k^j$ are the k trees returned by the parser for the sentence c^j and let $p_1^j \ldots p_k^j$ be their probabilities. t_i^j is replicated R_i^j times, where:

$$R_i^j = \text{round}\left(M * \frac{p_i^j}{\sum_{l=1}^{k} p_l^j}\right)$$

Finally, the size of the resulting treebank is $\sum_{j=1}^{N} \sum_{i=1}^{k} R_i^j$. Note that, under this definition, if the probability of a tree is too small, it will not be a part of the new

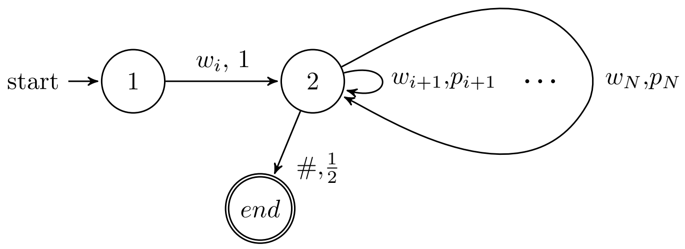


Figure 5.3: Typical meta-tree building automaton.

treebank at all. For computational reasons both k and M cannot be too large. In all our experiments, k and M are both set to 100.

TREEBANK GENERATOR**.** The aim of the TREEBANK GENERATOR is to build the first treebank that is given to the grammar inductor. This module uses *meta-trees*. A *meta-tree* of size n is a tree with a particular fixed structure of arcs, similar to a dependency tree, but with n variable nodes, where any sentence of length n can be fitted. These meta-trees have the particularity that their leaves are all different and that they are grouped by sentence length. Given an initial sentence of length n, a small treebank for this sentence is built via a two step procedure. First, all meta-trees whose yield has length n are selected and, second, all terminal symbols in the meta-trees are replaced by those in the sentence. The TREEBANK GENERATOR produces a new treebank by joining individual treebanks for all sentences. As a consequence, the resulting treebank contains the same trees for all sentences with the same length independently of the sentence words.

To generate the meta-trees that correspond to all n-long sentences, a special sentence $w_0, \ldots, w_n$, with all w_i different symbols, is processed by the parser and the tree replicator modules. The sentence is parsed with an *ad-hoc* PBG that we built specifically for each possible length. The *ad-hoc* PBG is defined by describing the automata for each word in the sentence. If w_i is the i-th terminal, then its right automaton is like the one shown in Figure 5.3. That is, the automaton has three states, one is final and absorbing, one is initial and has only one outgoing transition that is labeled with label w_i and probability 1. The third state is in between the two previous ones. It is connected to the final state by means of an arc labeled with probability 0.5 and label $\#$. Moreover, it has $n - i$ loops with labels $w_{i+1} \ldots w_n$

and probabilities p_j defined as:

$$p_j = \frac{\frac{1}{d_{ij}^s}}{2 * \sum_{d=1}^{n-1} \frac{1}{d^s}},$$

where d_{ij} is the distance between w_i and w_j, and the exponent s is a parameter in our model that modifies the mass of probability that are assigned to long dependencies. The three initializations, namely init-1, init-2 and init-3, we report in Section 5.4 are obtained by setting s to 1, 2 and 3 respectively. All p_i are such that their sum is not equal to 1, and in order to correctly define a probabilistic automaton, a fourth non-final and absorbing state is required to normalize the probability. The probability of going to this state is $1 - (0.5 + 0.5 * \sum_{l=i+1}^{m} p_l)$. This state is not shown in the picture. Intuitively, the probability of having many dependents and of having long distance dependents diminishes with an exponential factor s. The bigger the s the less likely are these two situations. The $2 * m$ automata, 2 per terminal symbol in the sentence, plus one automaton for the *root* symbol, are used to define a PBG G. Following the general schema, G is used to produce the k-best parsers of sentence $w_0, \ldots, w_m$. These k trees are fed into the TREEBANK GENERATOR, and it produces the treebank of meta-trees.

5.3 Automata Learning Algorithms

We use two different algorithms for learning probabilistic automata. First, we propose the Minimum Discrimination Information (MDI) algorithm (TDdlH00), which infers automata in a fully unsupervised fashion. It takes a bag of strings and returns a probabilistic deterministic automaton. Briefly, the MDI algorithm produces an automaton by first building a prefix tree that accepts only the training material. Moreover, the prefix tree contains one path for each string and it contains as many final states as there are different strings. All arcs in the prefix tree are marked with the number of times each arc is traversed while mapping strings into paths. These numbers are then transformed into probabilities which results in a probabilistic automaton that recognize exactly the training material. The algorithm proceeds to look for pairs of states that can be merged into one single state. Two states can be merged if the probability distribution defined by the automata that results from the merge is not too far away[1] from the distribution defined by the prefix tree. The

[1]The difference between the two probability distributions is computed by means of the Kullback-Leibler divergence(KL51).

algorithm proceeds greedily until no further merges can be done. MDI has only one parameter, α, that can be used to control the maximal allowed distance between two distributions before a merge is performed. α is a real number between 0 and 1; when it is equal to 0, no merges are allowed while when equal to 1 all merges are allowed. The MDI algorithm receives a bag of strings together with a value for the parameter α, and it outputs a probabilistic automaton.

Second, we contribute an *ad hoc* algorithm that only learns the transitions probabilities of a given automaton backbone structure. A backbone structure is a deterministic finite automaton without probabilities. It receives a bag of strings and a automaton backbone structure and returns the automaton backbone structure plus their probabilities. The backbones we use are general enough to warranty that they accept all strings in the training material. In contrast, our second algorithm is not fully unsupervised because it receives, along with a bag of strings, the backbone of the automaton it should produce. The backbone consists of the states and the arcs of the automaton, but it is missing the transition probabilities. It is the task of our Given Structure (GS) algorithm to find them. As we see in Section 5.5, DMV and EVG define a particular skeleton to their automata and as is the GS, the skeleton is information that is given to the algorithm as prior knowledge. In this sense, MDI is fairer learner than GS given that is works with less information. In our experiments we show that even with less information, MDI works better than GS. We currently experiment with different skeletons, but all of them have similar structure: They have a unique absorbing final state, and N intermediate states $S_1 \dots S_N$. The skeleton has one arc between states S_i and S_{i+1} for each possible label, and one arc between states S_i and the final state, labeled with the end of production symbol $\#$. The GS algorithm uses the training material to estimate all arcs probabilities. Since the skeleton is both deterministic and expressive enough, there is a path in the automaton for each sequence in the training material. The GS algorithm maps each sentence to a path in the automaton, it records the number of times each arc has been used, and, finally, it transforms those counters into probabilities. GS-N refers to the GS algorithm when it uses a backbone with N states as describe above.

For example, with $N = 3$, and the following training material:

$$\{\texttt{NN \#,NN VB \#,NN NN \#,NN VB NN \#}\}$$

Figure 5.4 (a) shows the skeleton with the corresponding weights (zero weights arcs have been removed), and Figure 5.4 (b) shows the same automata with weights transformed into probabilities.

We use these skeletons because they are the easiest structure that can be manu-

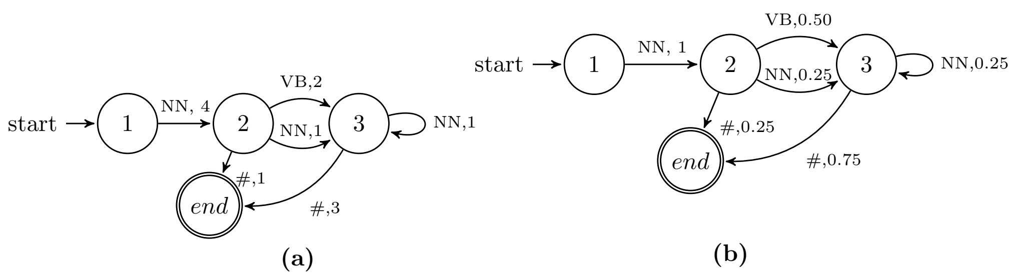


Figure 5.4: An example of a GS-3 automaton. (a) The skeleton displaying the number of times each arc has been used. Arcs with zero weight are not shown. (b) The result of transforming weights into probabilities.

ally described without making strong assumptions about the underlying language. We experiment with two different skeletons both having 3 and 4 states respectively. The skeleton with 3 states pays special attention to the first dependent while the one with 4 to the first two dependents. Moreover, GS-3 automata generate dependents independently of their siblings. GS-4 automata can recall if a dependent is the first one or not. In general the GS-i can recall if there has been less that $i - 1$ dependents. Our GS algorithm is also a generalization over n-grams which can be seen as instances of GS-n where the skeleton has a particular shape. Moreover, Section 5.5 shows that they can be seen as instances of a version of GS that allows non-deterministic backbones.

5.4 Experimental Results

Our model was tested on sentences with a restricted length up to 10 POS. Due to computational costs, the current version of our parser cannot deal with long sentences. However, we have some experiments which use sentences with up to 20 POS with promising results.

We report results for English, German, Swedish, Turkish, Bulgarian, Spanish, Portuguese and Danish. We tested 3 different initializations in the TREEBANK GENERATOR module, and two different algorithms for learning automata in the PBGs INDUCTOR module. This showcases the flexibility of our framework . All of our experiments used syntactic categories —POS tags— instead of words. The English model was induced using sentences in the Penn treebank (PTB) (MS93) with at most ten words (usually called WSJ10 (Kle05, HJM09, SE05)). Sections 2 to 21, that is, 6,007 sentences in total, were used for training. Testing was done using the 398 sentences of Section 23. Table 5.1 compares our English results with others in the literature. We report percentage of good attachment both directed and undirected. From the table, it can be seen that our results are comparable with the state-of-the-art ones, even for lexicalized instances. Our best result is what we call MDI-0: MDI with $\alpha = 0$ using init-2.

Our best performing models are those that can model the language of dependents as finite languages. Moreover, an inspection on the resulting automata shows that all models tend to create very short sequence of dependents, mostly of length up to one. To better understand our results, it is important to think our learning algorithm as a two-step process. First, the parser and the replicator define the training material that is going to be given to the automata learning algorithms. In a second phase, all the automata are learned. It is interesting to note that both the MDI and the GS-n algorithms generalize less over the training material as their respective parameters α and n go to zero and ∞, respectively. The MDI algorithm ends up building a tree like automata that recall all and only those strings in the training material. The probability assigned to each string is proportional to the number of times it occurs in the training material. In contrast, a GS-n automaton recalls the number of times each tag occurs as the i-th dependent. In this sense, MDI-0 generalizes less than any GS-n. In both cases, the resulting automaton accepts only finite languages.

From the experiments, it is clear that GS-4 improves over GS-3, and both EVG and DMV. It is surprising that our models obtain good results without resorting to smoothing which is usually applied in other models. As the experiments show, good results are obtained with initializations that penalize long distance dependencies and a high number of dependents. In other words, the models that work better are those that use initialization where words have fewer dependents and where dependents are close to their heads. If we compare the automata that results at the end of our algorithm, when init-2 and init-3 is used, the most noticeable feature is that, even for GS-3 and GS-4, the probabilities associated to cycling arcs are zero or very close to zero. When init-1 is used, cycles occur in the final automata of GS-3 but only a few in GS-4. Note that MDI-0 is stable across different initializations. If we look at the resulting automata, they all accept finite languages and moreover, all elements in the languages contain only a few symbols per string.

Since there are many parameters in our setup, it might be the case that our models are optimized for English. To validate our model, and unsupervised models in general, it is important to test their performance also in languages other than English. Table 5.3 compares our results for other languages. We compare them against standard baselines like right and left attachment, DMV, and the best results

model	accuracy
Attach-Right (KM04)	33.4
DMV-standard (KM04)	45.8
DMV-babysteps (@15) (SAJ10a)	55.5
DMV-babysteps (@45) (SAJ10a)	55.1
DMV-diriclet (CGS08)	45.9
Log-Normal Families (CGS08)	59.4
Shared Log-Normals (tie-verb-noun) (CS09)	61.3
Bilingual Log-Normals (tie-verb-noun) (CS09)	62.0
EVG-Smoothed (skip-head) (HJM09)	65.0
EVG-Smoothed (skip-val) (HJM09)	62.1
Viterbi EM (SAJM10)	**65.3**
EVG-Smoothed (skip-head), Lexicalized (HJM09)	68.8
Hypertext Markup (SAJ10b)	**69.3**
LexTSG-DMV $(P_{lcfg}, P_{cfg}, P_{sh})$ (BC10)	67.7
our model (parameters)	accuracy
MDI, $\alpha = 0$, init-1	67.4
MDI, $\alpha = 0$, init-2	**69.0**
MDI, $\alpha = 0$, init-3	67.1
GS-3, init-1	50.7
GS-3, init-2	66.5
GS-3, init-3	67.0
GS-4, init-1	55.5
GS-4, init-2	66.7
GS-4, init-3	**67.6**

Table 5.1: Directed accuracies on Section 23 of WSJ10 for several baselines and recent systems.

Lang.	#sen.	#POS	Lang.	#sen.	#POS
English	6007	36	German	12089	51
Turkish	3203	28	Swedish	3255	40
Bulgarian	5713	40	Portuguese	2406	19
Danish	1757	24	Spanish	595	23

Table 5.2: size of the training corpus for each language and the number of differents POS tags.

reported in results reported in (GGG$^+$10). According to (GGG$^+$10), German and Turkish best results are obtained by one model while the score for English and Swedish by two other different models. The fourth row display the highest score independently of the model used to obtain it. All corpora were part of the ConNLL-X special task on parsing (BM06). We show results using treebanks for Swedish (NHN05), German (BDH$^+$02), Turkish (OSHTT03, AOS03), Bulgarian (SOS$^+$02), Spanish (CM04), Portuguese (ABHS01) and Danish (KML03). Trees that were non-projective or that had more that one root were discarded as well as all trees whose sentences were longer that 10 words. Except Turkish, where the best performing model is the right attach baseline, all instances of our algorithm improve over DMV and the baselines.

Figure 5.6 and Figure 5.7 show the evolution of the directed accuracy and logarithmic likelihood respectively. Figure 5.6 shows, for each language, the directed accuracy in the 30 first iterations measured against the gold trees from the training material. More specifically, while the X axe vary in the number of iteration, the Y axe plots, the directed accuracy of the trees that consists only of the most probably trees returned by the k-BEST PBG PARSER for each sentence in the training material[1]. For iteration number 0 we use the treebank returned by TREEBANK GENERATOR instead of k-BEST PBG PARSER.

Similarly, in Figure 5.7 we plot the logarithmic likelihood for each treebank in the first 30 iterations. It is important to remark that the gold trees are used only for analysis purposes, and they are not used inside the algorithm.

Two variables must be taken into account to decide which parameterization of our system should be used for a given language: the number of sentences available for training and the number of POS tags.[2] Table 5.2 shows the variables for the

[1]This treebank is the same that produces as a result the 1-BEST PBG PARSER.

[2]More tags means a larger number of automata to be induced and thus less training material for each automaton.

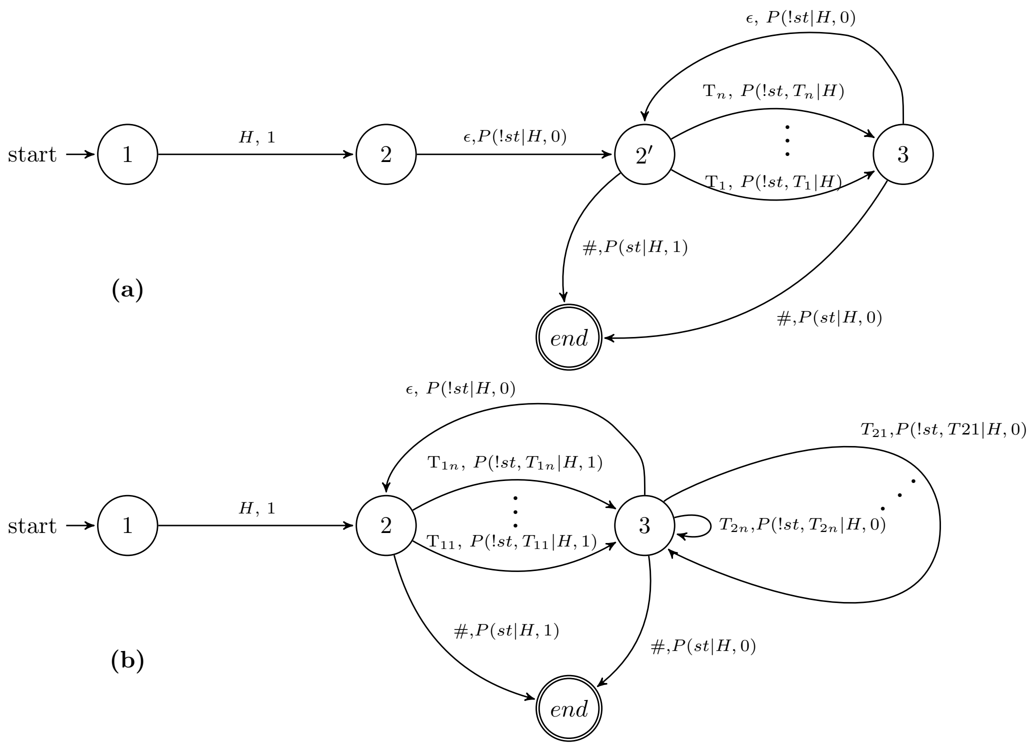


Figure 5.5: DMV (a) and EVG (b) automata.

model	English	German	Turkish	Swedish	Bulgarian	Spanish	Portuguese	Danish
(GGG$^+$10) DMV results	45.8/-	35.7 / -	46.8/-	39.4 / -	37.8 / -	40.3/-	35.7/-	47.2/-
left attach	24.1/54.6	25.9 / 53.2	5.1 / 54.8	28.5 / 56.3	40.5 / 59.9	29.0 / 55.2	34.1 / 61.7	43.7 / 60.1
right attach	33.4 / 56.3	29.0 / 52.1	**63.8 / 68.5**	28.5 / 55.5	20.2 / 56.3	29.4 / 55.2	27.9 / 55.5	17.2 / 57.5
(GGG$^+$10) best result	64.4 / -	47.4 / -	56.9 / -	**48.6 / -**	**59.8 / -**	**62.4 / -**	54.3 / -	46.6 / -
GS-3 Init-1	50.7 / 64.9	48.4 / 60.3	52.6 / 65.1	45.8 / 59.8	48.6 / 63.9	57.1 / 68.2	55.4 / 66.2	36.6 / 59.4
GS-3 Init-2	66.5 / 72.1	49 / 60.4	20.3 / 53.8	47.4 / 61.1	48.3 / 63.6	45.4 / 63.3	39.3 / 62.9	**47.8 / 66.2**
GS-3 Init-3	67.0 / 71.5	46.5 / 59.3	20.2 / 53.4	41.5 / 58.3	33.8 / 54.9	38.2 / 58.9	37.9 / 61.7	44.2 / 62.8
GS-4 Init-1	55.5 / 66.6	48.4 / 60.8	53.0 / 65.3	46.2 / 60.2	34.4 / 55.3	55.2 / 66.8	**55.6 / 66.6**	38.9 / 59.6
GS-4 Init-2	66.7 / 72.2	48.5 / 60.4	43.2 / 60.2	47.5 / 61.1	47.5 / 63.0	44.9 / 63.3	39.3 / 62.9	41.5 / 60.5
GS-4 Init-3	67.6 / 71.8	47.8 / 60	25.4 / 53.5	42.5 / 59.2	48.6 / 63.9	38.2 / 58.9	37.9 / 61.7	43.0 / 63.5
MDI-0 Init-1	67.4 / 72.4	47.7 / 59.9	52.4 / 64.8	45.4 / 59.3	35.8 / 55.5	51.0 / 62.4	49.4 / 63.5	35.3 / 57.7
MDI-0 Init-2	**69.0 / 73.3**	**54.1 / 63.3**	38.4 / 58.2	**48.1 / 61.3**	55.0 / 68.6	48.7 / 64.6	30.6 / 55.5	44.1 / 64.2
MDI-0 Init-3	67.2 / 72.4	53.8 / 63.3	24.5 / 53.0	46.2 / 60.7	38.1 / 56.5	46.0 / 64.0	30.8 / 55.8	44.7 / 65.0

Table 5.3: Our results expressed in (directed/undirected) accuracy for a variety of languages compared with the baselines: right attach, left attach and standard DMV results. We also report the state-of-the-art results for these languages.

languages used in this work. Table 5.3 shows that MDI-0 is very robust for those languages that have available a corpus with a significant number of sentences. This is the case of languages such as English, German and Bulgarian. For languages with a reduced number of sentences or a big number of POS, we should use GS-3 or GS-4. Intuitively, if we have less training material, models like GS-3 or GS-4 perform better because they generalize over the training material better than MDI-0.

where d_{ij} is the distance between w_i and w_j, and the exponent s is a parameter in our model that modifies the mass of probability that are assigned to long dependencies. The three initializations, namely init-1, init-2 and init-3, we report in Section 5.4 are obtained by setting s to 1, 2 and 3 respectively. All p_i are such that their sum is not equal to 1, and in order to correctly define a probabilistic automaton, a fourth non-final and absorbing state is required to normalize the probability. The probability of going to this state is $1 - (0.5 + 0.5 * \sum_{l=i+1}^{m} p_l)$. This state is not shown in the

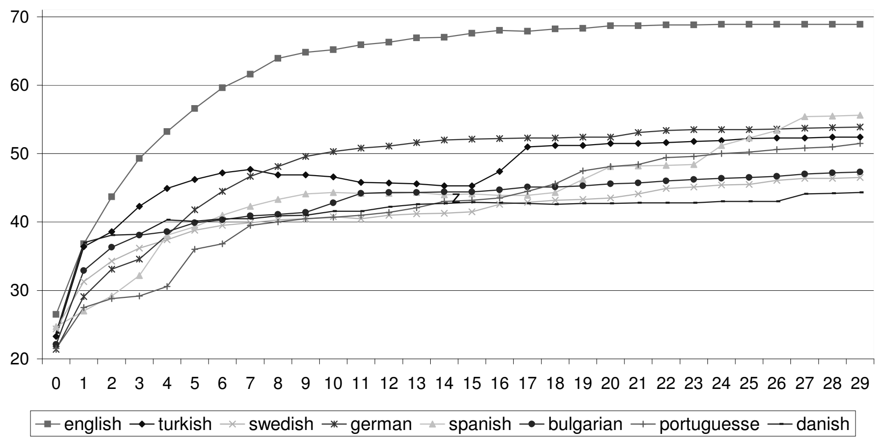


Figure 5.6: Directed Accuracy evaluated for each language over the first 30 iterations of our parser. The (Y) axe plots the directed accuracy calculated over the treebank obtained in the (X) axe iteration against the gold trees of the training material.

picture. Intuitively, the probability of having many dependents and of having long distance dependents diminishes with an exponential factor s. The bigger the s the less likely are these two situations. The $2 * m$ automata, 2 per terminal symbol in the sentence, pl

5.5 Our Model in Perspective

Most unsupervised approaches to unsupervised parsing are based on Dependency Model with Valence (DMV). DMV implements an EM algorithm that maximizes the likelihood of a particular grammar. This grammar can be seen as PBG where all its automata are like the one in Figure 5.5 (a). The probability between states 1 and 2 is the probability of generating a particular head. The one between states 2 and $2'$ is the probability of generating any dependent using a ϵ movement; the one between states 2 and end is the probability of not generating any dependent; the one between $2'$ and 3 is the probability to generate a particular dependent with its corresponding probability, the one between 3 and $2'$ is the probability of generating a new dependent, modeled again with an ϵ move, and finally, the probability between 3 and end is the probability of stop generating. In general, is not possible to transform a non-deterministic automaton to a deterministic one (DDE05). But for this particular case, the automata can be transformed to one without ϵ moves, but having in mind that some of the its arc probabilities are correlated and conse-

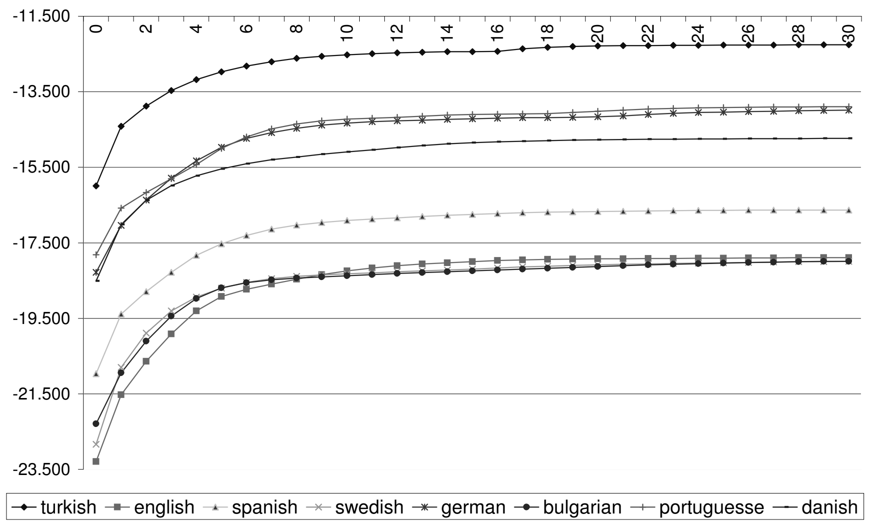


Figure 5.7: Evolution of logarithmic likelihood for each language. We evaluated over the treebanks induced in the first 30 iterations of our parser.

quently can not be learned independently. Smith and Eisner Cohen et al. (CGS08) derive a Variational Bayes EM for the DMV model. They investigate two priors for the variational Bayes (VB) algorithm, the Dirichlet and the Logistic Normal prior. They also initialize this variant of EM with the Klein and Manning initialization. Their best results were obtained with the Logistic Normal distribution, it was 59.3 of directed accuracy.

Spitkovsky et al. (SAJ10a) use the DMV model, but they introduce two interesting techniques. First, they use an incremental initialization that starts with sentences of length 1, and only later uses longer sentences. Second, they analyze the trade-off between complexity and quality in the training phase. They found that training with sentences up to length 15 performs better than training with longer sentences when testing in section 23 of WSJ10, WSJ20, WSJ30, WSJ100 and WSJ∞, among others.

Headden et al. (HJM09) extend DMV by adding a parameter that distinguishes the probabilities of the first dependent from the probabilities of the subsequent ones. As in the DMV, even with the automata not being explicitly defined, they can be re-built from the definition of the model. Figure 5.5 (b) shows such an automaton. The probability between states 1 and 2 is the probability of generating a particular head, between 2 and 3, are the probabilities of generating a particular dependent as the first one, between 3 and 3 are the probabilities of generating a dependent that is not the first one anymore, the probability between 2 and end is the probability of not having any dependents, and finally the probability between 3 and end is the probability of stopping generating dependents. To maximize the likelihood of their model they use a Variational Bayes EM algorithm with a Dirichlet prior (similar to (CGS08)) and they used a linearly smoothed model to deal with the data sparseness. As their initialization, they randomly sample some sets of trees and choose the best ones using some iterations of a Variational Bayes EM algorithm. They show that, by including smoothing, they improve over DMV obtaining the best result that is known for unsupervised parsing. The unsmoothed version of the EVG model corresponds exactly to our GS-3 model. The DMV model without the one-side-first parameter, is in between GS-2 and GS-3. It does not distinguish the probabilities for the dependent generation, as in GS-2, but the probability of stopping is distinguished like in GS-3. Gillenwater et al. (GGG^{+}10) used a posterior regularization (PR) framework (GT07) instead of the traditional EM algorithm. They model their dependencies as in DMV, and as variants of the EVG model. They argue that the main problem with unsupervised parsing is data sparseness and their model deals with such problem adding constraint that control for long dependencies. They report substantial gains over the standard EM algorithm, but they are not stable across languages. Finally, our weak-EM corresponds to a full EM algorithm (Pre01) when a k-BEST PBG PARSER with $k = \infty$ is used. Full EM is not feasible in our setup because a ∞-best parsing requires an exponential amount of space and time.

Spitkovsky et. al. (SAJM10) is in one sense, the work most similar to ours, as we are also estimating the probabilities of a model given the previous model, albeit using k-best parse trees. They obtain good scores 44.8% for English, in long sentences (all sentences in section 23 of PTB).

Blunsom and Cohn (BC10) replace the simple underlying grammar commonly used by a probabilistic tree substitution grammar. This formalism is capable of better representing complex linguistic structures because they can learn long dependencies. To limit the model's complexity they used a Bayesian non-parametric prior. They obtained state-of-the-art results for English in long sentences 55.7%.

Using standard DMV, Spitkovsky et al. (SAJ10b) use Web mark-up for improving parsing up to 50.4% of directed accuracy.

5.6 Future Work

One of the most important aspects to continue our work is to extend our experiments to longer sentences. To do so, we should optimize our implementation of the parser to make it parallel. We performed some experiments with wsj15 and we obtained promising results, about 49% of directed accuracy, which is close to the state-of-the-art ones, about 53%.

Another experiment that we will perform is to use ideas from Chapter 3 in our dependency parser. Fully lexicalized models may be costly and too sensitive to data sparseness, nevertheless we think unsupervised parsers can benefit of a more appropiate granularity of POS tag sets. This idea may be implemented by selecting a POS tag and splitting the words with this POS by using a chosen feature function. For example, by selecting the POS `VB`, and splitting it using the distance of the word to the root node. We think of applying the split of POS tags starting with the initial tree-bank, which is obtained as in section 5.2. This split should be recalculated in each step of our learning architecture, after the new set of dependency trees is calculated by using the K-best parser. We hope that this idea may help to obtain more accurate dependency trees, specially with longer sentences because it could distinguish more complex dependency relationships by using more automata specialized according to the dependency languages of each POS tag considered.

Finally, our model allows us to choose different kinds of automata structures. Another interesting idea to improve this parsing model is to benefit from this flexibility of our model. An interesting experiment is to choose the automaton structure to be associated with a given POS tag according with the size of its training set. As the results obtained for different languages suggest, the automata structure[1] can be adapted to the size of dependency the tree-bank; our idea is to investigate potential relationships between the size of the training set associated with each POS tag.

5.7 Discussion and Conclusions

Over the last years NLP research have been focused in unsupervised dependency parsing, specially after that Klein presented the DMV parser (KM04). Most of the newest parsers like (CGS08, CS09, HJM09, GGG^{+}10, BC10) are essentially the DMV model which uses a different bias function in each step of its EM algorithm definition. This bias function penalizes long dependencies of the utterances. This penalization is needed, as it is explained in (SAJ09), because DMV reserves too much probability mass for what might be unlikely productions at the beginning, and the classic EM is not enough to redistribute such probability mass across the parse trees.

[1]Recall that we use the same automata structure for all POS tags.

We chose to take a different approach. Our algorithm implements a weak-EM algorithm that instead of computing the probability distribution over the whole forest of trees, uses a tree-replicator module that builds tree-banks resembling the most likely area of the probability distribution. The resulting algorithm allows us to test different ways to model dependents and different ways to compute automata.

Instead of using a penalization in each iteration of the EM algorithm, in our model we use different biased tree-banks which perform the penalization of long dependencies. We show experimental results using three different initializations, two automata learning algorithms and eight different languages.

Our experiments showed that, for a given language, we have to choose a parameterization of our system that generalizes across different training sets depending on the size of training material available for this language. We show training size influences parameterization in a predictable manner.

Chapter 6

Conclusion

In NLP we try to build a language model that is a formal construct that gives account of the patterns appearing in the utterances of a language. Methods are developed in order to make explicit the hidden linguistic structure from examples of actual sentences of a given language.

Among the main formalisms used to describe natural languages structures, Context Free Grammars occupy a prominent place in spite of their limitations. Indeed, what characterizes Context Free Languages is the independence assumption of constituents with respect to their contexts. Other formalisms were proposed that seek to enrich CFGs with some annotations which add mild contextual information to the rules of the grammar. Dependency grammars do not make the notion of rule of generative grammars explicit, though a kind of Context-Free-like grammar can be easily obtained from a dependency one. Moreover, these grammars make explicit the dependency relations between words.

One formalism for dependency grammars is the so called Probabilistic Bilexical Grammars. This model assumes that the language of dependencies at right and left of each POS are regular. Given the results obtained in this work, BGs have proved to be useful and accurate for the study and representation of dependencies in language. Indeed, we used them as a formalism all across our work and obtained state-of-the-art results in different parsing models (Sections 3.5.3, 4.7 and 5.4).

Data-driven methods attempt to address the problem of language modeling using two different approaches: so-called supervised methods incorporate as input the result of human-intensive work in the form of linguistic analysis and annotations while unsupervised methods look for the syntactic structures in the bare sentences in a given language.

Naive models obtained by simply assuming that the phenomena found in a large enough sample of data are exact representatives of the model of the whole language don't generalize too well to unseen data. On the other hand, heuristic methods based on aprioristic intuition about the structure of the language may lack empirical evidence when applied to real data. In this work we developed algorithms that automatically search linguistic patterns.

In Chapter 3, we addressed the issue of granularity of the POS-tag set and described a method as well as a measure that can be used to find potential splitting or merging of POS-tags. The criterions for optimallity of POS-tag sets are strongly task dependent, but our method can be used to improve parsing as shown in Section 3.5.3. The measure we use, and some others we propose are combinations of statistical data and "intrinsic" properties of the automata. In fact, words are tagged depending of the expected contexts in which they appear and this tagging creates a context for accompanying words to be tagged. Appendix A shows different splittings of POS-tags, according which information is implicitly encoded in tags. As this information is obtained from dependencies, we expect higher improvements in constituent based parsers.

While looking for optimizations of head finder rules, our findings of Section 4.7 confirm those by Bikel (Bik04b) in the sense that their presence is crucial for the performances of dependency parsers but their variations are not. The reason lies in that the importance of head finding is the passing of context information, which is actually what improves the behavior of parsers, rather than rules definitions and variations. Indeed, head finding algorithms look and search words belonging to special grammatical classes in order to determine where the head of a constituent lies. So, the marking of heads amounts to pass information obtained from contexts. Which information should be considered more or less relevant is to a large extent task dependent and a universal best method of head finding might make little sense. We perform analysis of the individual impact of each rule and in Appendix B we list a number of improvements found by our algorithm to Magerman original rules. We also found higher variance in Collins' parser than Stanford parser when head finding rules are changed. This fact can be attributed to the full lexicalization in Collins' parser that makes it more sensitive to variations.

We found that it is possible in to add useful information obtained in our research works to the existing supervised parsers. The results described in Sections 3.5.3 and 4.7 show that state-of-the-art parsers perform better both when the parser is fed with our new set of POS, and when the new set of head-rules is used.

Working on unsupervised dependency parsing, we found that state-of-the-art performances can be achieved using a simple and flexible method. This result is validated across a dozen languages. As shown in Section 5.4. As far as we know, with the exception of (GGG^{+}10), previous works on unsupervised parsing are based principally in English or some other single language. Cross language validation provides further evidence of the robustness of the methods applied. In fact, good behavior in different languages indicates that this unsupervised method can learn about regularities of different sorts as opposed to achieve good results in a language just because the method has an implicit bias for the features of this particular case.

The flexibility of our method can also accomodate to different sizes of training sets. We have freedom to choose the automata structure which best adapts to existing data (Section 5.4).

Finally, as some researchers state in their work, we think that while the NLP supervised methods for parsing are well-studied and well-understood, the unsupervised methods for parsing are only well-studied, and an open area of research. This fact explains why our results on supervised methods may have less impact; because their learning architectures are very optimized and tuned with a lot of features including contextual dependency information. As a consequence, the information that we provide to them hardly improves their performance.

In contrast, the results we achieved in unsupervised dependency parsing are comparable to the state-of-the art ones with a very simple, flexible and easily reproducible architecture.

6. CONCLUSION

Appendix A

New set of POS Tags Optimized

In this Appendix we include the different POS obtained in our research task on Chapter 3. Each table describes how the original PTB tag selected is split by our optimization procedure. Such process starts with an initial split and applies a given feature from Table 3.2.

new POS	Feature Value	new POS	Feature Value	new POS	Feature Value
NEWTAG_1	2	NEWTAG_4	10	NEWTAG_7	6,17,18,20
NEWTAG_2	4	NEWTAG_5	0,16,7	NEWTAG_8	7,11
NEWTAG_3	8	NEWTAG_6	1,13,9,12	NEWTAG_9	3,14,15

Table A.1: New tags related to `VB,MD` that were calculated with feature `Depth`.

new POS	Feature Value	new POS	Feature Value	new POS	Feature Value
NEWTAG_1	11,7,0,3	NEWTAG_4	19	NEWTAG_7	1,16
NEWTAG_2	9,12	NEWTAG_5	2	NEWTAG_8	17,18,20
NEWTAG_3	10	NEWTAG_6	5,13	NEWTAG_9	6,15
NEWTAG_10	8	NEWTAG_11	4,14		

Table A.2: New tags related to `VBN,VB,MD` that were calculated with feature `Depth`.

new POS	Feature Value	new POS	Feature Value
NEWTAG_1	VBN	NEWTAG_4	WRB, DT
NEWTAG_2	PRP	NEWTAG_5	PDT,CC,RBR,-LRB-,RB,NNPS,POS,-RRB-,JJS,RBS
NEWTAG_3	IN	NEWTAG_6	NNP,VBG
NEWTAG_7	MD,NN	NEWTAG_10	NNS
NEWTAG_8	WP$,VBZ	NEWTAG_11	VBD
NEWTAG_9	TO	NEWTAG_12	VB
NEWTAG_13	JJ	NEWTAG_16	WDT
NEWTAG_14	CD,NONE	NEWTAG_17	WP,JJR,$,ROOT
NEWTAG_15	VBP		

Table A.3: New tags related to `VB,MD` that were calculated with feature `gFather`.

new POS	Feature Value	new POS	Feature Value
NEWTAG_1	CD	NEWTAG_4	WP$
NEWTAG_2	PDT,CC,RBR,-LRB-,NNPSPOS,-RRB-,JJS,RBS	NEWTAG_5	PRP
NEWTAG_3	RP	NEWTAG_6	TO
NEWTAG_7	SYM	NEWTAG_10	WDT
NEWTAG_8	RB,NNS	NEWTAG_11	VBP
NEWTAG_9	VBN	NEWTAG_12	MD
NEWTAG_13	NN,NONE	NEWTAG_16	JJ
NEWTAG_14	VB	NEWTAG_17	VBD
NEWTAG_15	IN	NEWTAG_18	WP,$,ROOT
NEWTAG_19	NNP,VBG		
NEWTAG_20	WRB,DT		
NEWTAG_21	JJR		

Table A.4: New tags related to `VBN,VB,MD` that were calculated with feature `gFather`.

new POS	Feature Value	new POS	Feature Value	new POS	Feature Value
NEWTAG_1	2	NEWTAG_4	8	NEWTAG_7	6,12
NEWTAG_2	9	NEWTAG_5	1	NEWTAG_8	0,16,3,14
NEWTAG_3	13,4	NEWTAG_6	6,17,18,15,19	NEWTAG_9	11,7
NEWTAG_10	10,5				

Table A.5: New tags related to `VB,MD` that were calculated with feature `NumChanges`.

new POS	Feature Value	new POS	Feature Value	new POS	Feature Value
NEWTAG_1	20	NEWTAG_4	12	NEWTAG_7	5
NEWTAG_2	3	NEWTAG_5	11,7,16	NEWTAG_8	1
NEWTAG_3	9	NEWTAG_6	2	NEWTAG_9	17,18,19
NEWTAG_10	13,14	NEWTAG_13	8		
NEWTAG_11	4	NEWTAG_14	10		
NEWTAG_12	0				

Table A.6: New tags related to `VBN,VB,MD` that were calculated with feature `NumChanges`.

new POS	Feature Value	new POS	Feature Value	new POS	Feature Value
NEWTAG_1	8	NEWTAG_4	5	NEWTAG_7	6
NEWTAG_2	3	NEWTAG_5	0	NEWTAG_8	4,10
NEWTAG_3	1	NEWTAG_6	11,7,9,12,2,14,13		

Table A.7: New tags related to `VBN,VB,MD` that were calculated with feature `VerbAllDepth`.

new POS	Feature Value	new POS	Feature Value
NEWTAG_1	3	NEWTAG_4	5
NEWTAG_2	7,8,1	NEWTAG_5	9,12,2,10
NEWTAG_3	6,0	NEWTAG_6	4

Table A.8: New tags related to `VBN,VB,MD` that were calculated with feature `VerbDepth`.

new POS	Feature Value	new POS	Feature Value	new POS	Feature Value
NEWTAG_1	4	NEWTAG_4	9,8,0,10	NEWTAG_7	3
NEWTAG_2	1	NEWTAG_5	7		
NEWTAG_3	5	NEWTAG_6	6,2		

Table A.9: New tags related to `VBN,VB,MD` that were calculated with feature `VerbVBDepth`.

new POS	Feature Value	new POS	Feature Value	new POS	Feature Value
NEWTAG_1	15	NEWTAG_4	7,5	NEWTAG_7	6,11,9,12,13,10
NEWTAG_2	2	NEWTAG_5	4	NEWTAG_8	8,1
NEWTAG_3	3	NEWTAG_6	0		

Table A.10: New tags related to `VBN,VB,MD` that were calculated with feature `NumSib`.

new POS	Feature Value	new POS	Feature Value	new POS	Feature Value
NEWTAG_1	VBZ	NEWTAG_4	WDT	NEWTAG_7	DT
NEWTAG_2	NNPS	NEWTAG_5	EX	NEWTAG_8	:
NEWTAG_3	VBG	NEWTAG_6	WRB	NEWTAG_9	RBR
NEWTAG_10	MD	NEWTAG_13	VBP	NEWTAG_16	JJ,-RRB-
NEWTAG_11	CC,VBN	NEWTAG_14	JJR	NEWTAG_17	VBD
NEWTAG_12	JJS	NEWTAG_15	RP,NONE	NEWTAG_18	NN
NEWTAG_19	RB	NEWTAG_22	$	NEWTAG_24	IN
NEWTAG_20	VB	NEWTAG_23	CD	NEWTAG_25	TO
NEWTAG_21	WP			NEWTAG_26	NNP
NEWTAG_28	,	NEWTAG_27	FW,POS,NNS,RBS,PRP$		
NEWTAG_29	-LRB-,PRP				

Table A.11: New tags related to `VB` that were calculated with feature `FstRightDep`.

new POS	Feature Value	new POS	Feature Value	new POS	Feature Value
NEWTAG_1	(7,3),(13,7)	NEWTAG_4	(20,4)	NEWTAG_7	(16,6),(16,5)
NEWTAG_2	(17,5)	NEWTAG_5	(11,1)	NEWTAG_8	(15,8)
NEWTAG_3	(19,8)	NEWTAG_6	(5,2)	NEWTAG_9	(5,3)
NEWTAG_10	(11,3)	NEWTAG_13	(14,2)	NEWTAG_16	(8,4),(7,4)
NEWTAG_11	(8,3)	NEWTAG_14	(9,4)	NEWTAG_17	(3,2),(0,0)
NEWTAG_12	(14,8)	NEWTAG_15	(13,6)	NEWTAG_18	(17,4),(5,4)
NEWTAG_19	(14,6),(6,5),(15,4),(15,3),(18,8),(20,10),(15,7),(11,7),(12,7),(10,7),(14,7),(5,0),(17,8),(18,6), (15,6), (10,2),(16,9),(7,0),(13,3),(14,3),(16,3),(18,5),(16,8),(17,6),(16,4),(20,9),(15,2),(8,0), (10,1),(8,1),(18,10),(9,1)				
NEWTAG_20	(9,3),(6,0),(16,7),(10,3),(4,0),(7,5),(1,0),(13,4)			NEWTAG_24	(17,2)
NEWTAG_21	(5,1),(9,2),(10,4),(2,0)			NEWTAG_25	(7,1)
NEWTAG_22	(4,1)	NEWTAG_23	(7,2),(6,3)	NEWTAG_26	(11,4)
NEWTAG_27	(3,0)	NEWTAG_30	(8,6),(12,5)	NEWTAG_33	(10,6),(3,1)
NEWTAG_28	(11,5),(14,5)	NEWTAG_31	(6,2)	NEWTAG_34	(18,4)
NEWTAG_29	(10,5),(12,2)	NEWTAG_32	(12,6)	NEWTAG_35	(6,1),(15,5)
NEWTAG_36	(9,5)	NEWTAG_39	(13,1)	NEWTAG_42	(13,5)
NEWTAG_37	(13,2)	NEWTAG_40	(11,2),(4,2),(9,6)	NEWTAG_43	(6,4),(8,2)
NEWTAG_38	(12,3),(12,4)	NEWTAG_41	(9,0)	NEWTAG_44	(11,6)
NEWTAG_45	(4,3)				
NEWTAG_46	(8,5),(2,1),(14,4)				

Table A.12: New tags related to `VBN,VB,MD -VBN,VB,MD` that were calculated with combined features `Depth-VerbVBDepth`.

new POS	Feature Value	new POS	Feature Value	new POS	Feature Value
NEWTAG_1	(12,TO)	NEWTAG_4	(5,TO)	NEWTAG_7	(10,NN)
NEWTAG_2	(3,VB)	NEWTAG_5	(14,IN)	NEWTAG_8	(5,WDT)
NEWTAG_3	(15,NN)	NEWTAG_6	(10,VBP)	NEWTAG_9	(4,JJS)
NEWTAG_10	(8,-LRB-),(9,-LRB-),(5,CD),(2,NNP),(17,NN),(4,CD),(2,VB),(20,JJ),(12,VBD),(3,JJS), (4,-LRB-),(10,WRB),(6,PDT),(3,JJR),(18,TO),(13,VBG),(5,JJS),(15,VBN),(2,WDT), (3,CD),(11,RBR),(3,RBR),(13,VBZ),(2,JJ),(7,DT),(15,VB),(12,JJ),(7,$),(13,MD), (2,WP),(14,JJ),(17,WP),(18,VB),(9,RBR),(13,IN),(13,WP$),(18,WDT),(15,VBZ), (17,VBN),(6,CC),(15,WDT),(18,NNS),(9,DT),(6,NNP),(13,TO),(4,CC),(5,CC), (5,NNPS),(10,RB),(2,VBG),(12,VBP),(2,RB),(6,WP$),(16,WDT),(14,TO),(8,DT), (12,RB),(11,VBD),(14,MD),(10,TO),(13,VBP),(5,RB),(12,MD),(15,IN),(13,VB), (7,RB),(3,RBS),(6,JJR),(10,JJ),(5,-LRB-),(2,VBN),(11,NNS),(12,NN),(12,DT), (9,$),(6,RB),(6,-LRB-),(13,JJ),(13,RB),(2,WRB),(6,DT),(6,$),(15,WP),(12,JJR), (11,VBG),(6,JJS),(7,CD),(12,WRB),(11,WRB),(11,VBP),(14,VBD),(15,VBD),(17,VB), (16,NN),(14,WDT),(12,VBZ),(16,VBN),(11,WP),(8,NNP),(12,RBR),(9,RB),(3,NNPS),(15,TO), (5,NNP),(3,$),(8,CC),(10,JJS),(10,WP$),(14,VBZ),(3,CC),(2,-RRB-),(16,JJ), (17,VBP),(18,VBG),(15,VBG),(16,NNS),(8,RB),(4,DT),(3,-LRB-),(13,WP),(10,MD), (15,NNS),(20,VB),(8,CD),(9,WP),(2,-LRB-),(7,CC),,(14,VBG),(5,POS),				
NEWTAG_11	(7,TO)	NEWTAG_14	(17,TO)	NEWTAG_17	(16,IN)
NEWTAG_12	(6,VBD)	NEWTAG_15	(18,IN)	NEWTAG_18	(9,NNS)
NEWTAG_13	(19,VBN)	NEWTAG_16	(8,VBP)	NEWTAG_19	(6,JJ)
NEWTAG_20	(2,VBD)	NEWTAG_23	(3,RB)	NEWTAG_26	(5,VB)
NEWTAG_21	(2,VBZ)	NEWTAG_24	(13,VBN)	NEWTAG_27	(5,VBD)
NEWTAG_22	(9,VBZ)	NEWTAG_25	(3,NNP)	NEWTAG_28	(4,MD)
NEWTAG_29	(3,VBP),(10,NNS)	NEWTAG_32	(12,VB)	NEWTAG_35	(2,NNPS)
NEWTAG_30	(17,VBZ)	NEWTAG_33	(12,NNP)	NEWTAG_36	(6,VBZ)
NEWTAG_31	(7,VBG),(4,VBP)	NEWTAG_34	(7,WDT)	NEWTAG_37	(4,TO)

NEWTAG_38	(14,NNP)	NEWTAG_41	(4,NNS)	NEWTAG_44	(7,VBZ)
NEWTAG_39	(20,WP)	NEWTAG_42	(2,TO)	NEWTAG_45	(9,MD)
NEWTAG_40	(12,IN),(5,MD)	NEWTAG_43	(6,RBR)	NEWTAG_46	(16,WP)
NEWTAG_47	(17,VBG),(12,VBN)	NEWTAG_50	(3,IN)	NEWTAG_53	(9,VB)
NEWTAG_48	(6,TO),(11,TO)	NEWTAG_51	(13,VBD)	NEWTAG_54	(4,RBR)
NEWTAG_49	(13,WRB),(10,WDT)	NEWTAG_52	(8,VBZ)	NEWTAG_55	(4,IN)
NEWTAG_56	(15,MD)	NEWTAG_59	(10,IN)	NEWTAG_62	(14,WRB)
NEWTAG_57	(3,POS)	NEWTAG_60	(6,NNPS)	NEWTAG_63	(3,PRP)
NEWTAG_58	(2,SYM)	NEWTAG_61	(4,VBD)	NEWTAG_64	(4,NNP)
NEWTAG_65	(10,VBZ)	NEWTAG_68	(11,WP$)	NEWTAG_71	(14,NN)
NEWTAG_66	(5,VBN)	NEWTAG_69	(12,VBG),(9,VBN),(4,RB)		
NEWTAG_67	(5,WRB)	NEWTAG_70	(4,$),(12,NNS),(7,VBD),(14,VBN)		
NEWTAG_72	(3,VBG),(14,VB)	NEWTAG_75	(11,NN),(7,VB),(3,TO)		
NEWTAG_73	(4,VBG)	NEWTAG_78	(10,VBG)	NEWTAG_76	(6,VBG)
NEWTAG_74	(4,PDT)	NEWTAG_79	(8,TO)	NEWTAG_77	(5,WP$)
NEWTAG_83	(3,NN),(2,NN)	NEWTAG_86	(3,VBZ)	NEWTAG_89	(7,-LRB-)
NEWTAG_84	(6,IN)	NEWTAG_87	(4,NNPS)	NEWTAG_90	(9,JJR)
NEWTAG_85	(5,$)	NEWTAG_88	(7,VBP)	NEWTAG_91	(8,NNS)
NEWTAG_92	(17,IN)	NEWTAG_95	(3,MD)	NEWTAG_98	(6,VB)
NEWTAG_93	(3,WP)	NEWTAG_96	(11,VBZ)	NEWTAG_99	(8,NNPS)
NEWTAG_94	(2,IN)	NEWTAG_97	(17,VBD)	NEWTAG_100	(6,CD)
NEWTAG_100	(16,VBG)	NEWTAG_102	(4,WRB)	NEWTAG_105	(16,TO)
NEWTAG_101	(6,NN)	NEWTAG_103	(16,VBZ)	NEWTAG_106	(13,WDT)
NEWTAG_102	(12,WDT)	NEWTAG_104	(6,VBN)	NEWTAG_107	(4,VBZ)
NEWTAG_108	(2,$)	NEWTAG_111	(5,VBZ)	NEWTAG_114	(4,NN)
NEWTAG_109	(9,NN)	NEWTAG_112	(11,JJ)	NEWTAG_115	(10,RP)
NEWTAG_110	(3,DT)	NEWTAG_113	(7,NNP)	NEWTAG_116	(8,VBN)
NEWTAG_117	(9,TO)	NEWTAG_120	(3,VBD)	NEWTAG_123	(18,MD)
NEWTAG_118	(6,MD)	NEWTAG_121	(3,NNS)	NEWTAG_124	(10,NNP)
NEWTAG_119	(7,WP),(5,NNS)	NEWTAG_122	(5,DT)	NEWTAG_125	(9,VBP)
NEWTAG_126	(4,WDT)	NEWTAG_129	(9,VBD)	NEWTAG_132	
NEWTAG_127	(7,IN)	NEWTAG_130	(10,VBD)	NEWTAG_133	
NEWTAG_128	(8,VBG)	NEWTAG_131	(7,VBN),(9,WRB),(3,VBN),(8,WDT)		
NEWTAG_134	(5,JJR)	NEWTAG_137	(10,$)	NEWTAG_140	(10,VBN)
NEWTAG_135	(8,WRB)	NEWTAG_138	(8,IN)	NEWTAG_141	(8,VB)
NEWTAG_136	(11,RB)	NEWTAG_139	(4,VB)	NEWTAG_142	(16,VBP)
NEWTAG_143	(4,JJ),(10,VB)	NEWTAG_146	(9,WP$)	NEWTAG_149	(11,VB)
NEWTAG_144	(2,VBP),(8,JJ)	NEWTAG_147	(9,NNP)	NEWTAG_150	(13,NNP)
NEWTAG_145	(6,WRB),(7,MD)	NEWTAG_148	(14,WP)	NEWTAG_151	(8,WP$)
NEWTAG_152	(8,NN),(4,VBN)	NEWTAG_155		NEWTAG_158	(14,NNS)
NEWTAG_153	(11,VBN),(5,IN)	NEWTAG_156		NEWTAG_159	(5,NN)
NEWTAG_154	(3,WRB)	NEWTAG_157	(10,DT)	NEWTAG_160	(5,WP)
NEWTAG_161	(0,NONE),(12,WP),(7,NN)	NEWTAG_164		NEWTAG_167	(9,IN)
NEWTAG_162	(7,WP$)	NEWTAG_165	(8,JJR)	NEWTAG_168	(3,JJ)
NEWTAG_163	(11,NNP)	NEWTAG_166	(5,VBP),(13,NNS),(9,VBG),(13,NN)		
NEWTAG_169	(8,$)	NEWTAG_172	(5,RBR)	NEWTAG_175	(7,WRB)
NEWTAG_170	(6,VBP)	NEWTAG_173	(3,RP)	NEWTAG_176	(7,NNS)
NEWTAG_171	(9,JJ),(5,VBG)	NEWTAG_174	(8,RP)	NEWTAG_177	(5,JJ)
NEWTAG_178	(4,JJR),(10,JJR)	NEWTAG_181	,(6,WDT)	NEWTAG_183	(7,JJR)
NEWTAG_179	(11,MD),(11,WDT),(6,WP),(11,IN),(8,WP),(4,WP),(10,WP),(6,NNS),(3,WDT),(1,ROOT)				
NEWTAG_180	(2,NNS),(8,VBD)	NEWTAG_182	(2,MD),(4,WP$),(9,WDT),(8,MD)		

Table A.13: New tags related to `VBN,VB,MD -VBN,VB,MD` that were calculated with combined features `Depth- gFather`.

Appendix B

Optimized Head Rules

In this appendix we show the set of optimized head rules obtained with the optimization procedure described in Chapter 4. Each table contains the original Magerman-Collins (Mag95b, Col97) (original) and the optimized head rule (optimized) for each grammatical categorie.

original	`(ADJP (l NNS) (l QP) (l NN) (l $) (l ADVP) (l JJ) (l VBN) (l VBG) (l ADJP) (l JJR) (l NP) (l JJS) (l DT) (l FW) (l RBR) (l RBS) (l SBAR) (l RB))`
optimized	`(ADJP (l NP) (l NN) (l NNS) (r JJR) (l RBR) (l SBAR) (r JJ) (l JJS) (r RB) (r ADVP) (l VBN) (r VBG) (r RBS) (l QP) (r DT) (r FW) (l $) (r ADJP))`

Figure B.1: Original and optimized Head Rule for `ADJP`.

original	`(ADVP (r RB) (r RBR) (r RBS) (r FW) (r ADVP) (r TO) (r CD) (r JJR) (r JJ) (r IN) (r NP) (r JJS) (r NN))`
optimized	`(ADVP (l JJS) (l JJ) (l ADVP) (l RBR) (l CD) (r JJR) (l NP) (l IN) (r RB) (r FW) (r NN) (l TO) (r RBS) )`

Figure B.2: Original and optimized Head Rule for `ADVP`.

original	(CONJP (r CC) (r RB) (r IN))
optimized	(CONJP (l IN) (l RB) (r CC))

Figure B.3: Original and optimized Head Rule for CONJP.

original	(NAC (l NN) (l NNS) (l NNP) (l NNPS) (l NP) (l NAC) (l EX) (l $) (l CD) (l QP) (l PRP) (l VBG) (l JJ) (l JJS) (l JJR) (l ADJP) (l FW))
optimized	(NAC (l JJS) (r NNP) (l CD) (r NAC) (r VBG) (r ADJP) (l QP) (l NNS) (l FW) (r NP) (l EX) (l JJR) (l NNPS) (r JJ) (l NN) (r PRP) (l $))

Figure B.4: Original and optimized Head Rule for NAC.

original	(PP (r IN) (r TO) (r VBG) (r VBN) (r RP) (r FW))
optimized	(PP (r TO) (r VBG) (l IN) (l VBN) (r FW) (r RP))

Figure B.5: Original and optimized Head Rule for PP.

original	(QP (l $) (l IN) (l NNS) (l NN) (l JJ) (l RB) (l DT) (l CD) (l NCD) (l QP) (l JJR) (l JJS))
optimized	(QP (l IN) (r JJ) (r RB) (r NN) (l DT) (l NCD) (l $) (l NNS) (r CD) (l JJS) (r QP) (r JJR))

Figure B.6: Original and optimized Head Rule for QP.

original	(RRC (r VP) (r NP) (r ADVP) (r ADJP) (r PP))
optimized	(RRC (r ADJP) (l NP) (l ADVP) (r PP) (l VP))

Figure B.7: Original and optimized Head Rule for RRC.

original	`(S (l TO) (l IN) (l VP) (l S) (l SBAR) (l ADJP) (l UCP) (l NP))`
optimized	`(S (l S) (r ADJP) (l NP) (r TO) (r VP) (r UCP) (l SBAR) (r IN) )`

Figure B.8: Original and optimized Head Rule for `S`.

original	`(SBAR (l WHNP) (l WHPP) (l WHADVP) (l WHADJP) (l IN) (l DT) (l S) (l SQ) (l SINV) (l SBAR) (l FRAG))`
optimized	`(SBAR (r SBAR) (r WHADVP) (l SINV) (l SQ) (l FRAG) (r IN) (l WHADJP) (r DT) (l S) (l WHPP) (r WHNP) )`

Figure B.9: Original and optimized Head Rule for `SBAR`.

original	`(SBARQ (l SQ) (l S) (l SINV) (l SBARQ) (l FRAG))`
optimized	`(SBARQ (l S) (l FRAG) (r SBARQ) (l SINV) (r SQ) )`

Figure B.10: Original and optimized Head Rule for `SBARQ`.

original	`(SINV (l VBZ) (l VBD) (l VBP) (l VB) (l MD) (l VP) (l S) (l SINV) (l ADJP) (l NP))`
optimized	`(SINV (r SINV) (l VP) (l VBP) (l VBZ) (r MD) (l VB) (r VBD) (r S) (r NP) (l ADJP) )`

Figure B.11: Original and optimized Head Rule for `SINV`.

original	`(SQ (l VBZ) (l VBD) (l VBP) (l VB) (l MD) (l VP) (l SQ))`
optimized	`(SQ (l VB) (l MD) (l VBD) (l VP) (r VBZ) (r VBP) (l SQ) )`

Figure B.12: Original and optimized Head Rule for `SQ`.

original	`(VP (l TO) (l VBD) (l VBN) (l MD) (l VBZ) (l VB) (l VBG) (l VBP) (l VP) (l ADJP) (l NN) (l NNS) (l NP))`
optimized	`(VP (r VBZ) (r MD) (l VB) (r VBN) (r VBD) (l NNS) (l VBP) (l TO) (l VBG) (l ADJP) (l VP) (l NP) (r NN) )`

Figure B.13: Original and optimized Head Rule for `VP`.

original	`(WHADJP (l CC) (l WRB) (l JJ) (l ADJP))`
optimized	`(WHADJP (l CC) (r JJ) (r WRB) (l ADJP) )`

Figure B.14: Original and optimized Head Rule for `WHADJP`.

original	`(WHADVP (r CC) (r WRB))`
optimized	`(WHADVP (r CC) (r WRB))`

Figure B.15: Original and optimized Head Rule for `WHADVP`.

original	`(WHNP (l WDT) (l WP) (l WP$) (l WHADJP) (l WHPP) (l WHNP))`
optimized	`(WHNP (l WHADJP) (l WHNP) (l WP) (r WP$) (r WDT) (l WHPP) )`

Figure B.16: Original and optimized Head Rule for `WHNP`.

original	`(WHPP (r IN) (r TO) (r FW))`
optimized	`(WHPP (l TO) (r IN) (l FW) )`

Figure B.17: Original and optimized Head Rule for `WHPP`.

Bibliography

[AB96] Hiyan Alshawi and Adam L. Buchsbaum. Head automata and bilingual tiling: Translation with minimal representations. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 167–176, 1996. 36, 37

[ABHS01] S. Afonso, E. Bick, R. Haber, and D. Santos. Floresta sinta(c)tica: a treebank for portuguese. In *Proceedings of International Conference on Language Resources and Evaluation (LREC 2002)*, pages 216–219, 2001. 95

[AOS03] N. B. Atalay, K. Oflazer, and B. Say. The annotation process in the Turkish treebank. In *Proceedings of the 4th Intern. Workshop on Linguistically Interpreteted Corpora (LINC)*, pages 117–120, 2003. 95

[Bak79] J. 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. 28

[BC10] P. Blunsom and T. Cohn. Unsupervised induction of tree substitution grammars for dependency parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP 2010)*, pages 1204–1213, October 2010. 6, 35, 94, 100, 101

[BDH$^+$02] S. Brants, S. Dipper, S. Hansen, W. Lezius, and G. Smith. The TIGER treebank. In *IEEE Journal for Transactions on Learning Technologies (TLT1)*, 2002. 25, 38, 95

[BHHH01] Alena Böhmová, Jan Hajič, Eva Hajičová, and Barbora Hladká. The prague dependency treebank: Three-level annotation scenario. In Anne Abeillé, editor, *Treebanks: Building and Using Syntactically Annotated Corpora*. Kluwer Academic Publishers, 2001. 38

[Bik02] Daniel M. Bikel. Design of a multi-lingual, parallel-processing statistical parsing engine. In *Proceedings of the second international conference on Human Language Technology Research*, HLT '02, pages 178–182. Morgan Kaufmann Publishers Inc., 2002. 11, 13

[Bik04a] D. Bikel. A distributional analysis of a lexicalized statistical parsing model. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP 2004)*, 2004. 39

[Bik04b] D. Bikel. *On the Parameter Space of Generative Lexicalized Statistical Parsing Models*. PhD thesis, University of Pennsylvania, 2004. 56, 71, 72, 73, 104

[Bik04c] Daniel M. Bikel. Intricacies of collins' parsing model. *COMPUTATIONAL LINGUISTICS*, 30:479–511, 2004. 40, 84

[BM06] S. Buchholz and E. Marsi. CoNLL-X shared task on multilingual dependency parsing. In *Conference on Computational Natural Language Learning (CoNLL-X)*. SIGNLL, 2006. 38, 95

[Bod06] Rens Bod. Unsupervised parsing with u-dop. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 85–92, Stroudsburg, PA, USA, 2006. 8

[BT73] T.L. Booth and R.A. Thompson. Applying probability measures to abstract languages. *IEEE Transactions on Computers*, C-22(5):442–450, 1973. 21

[CB02] D. Chiang and D. Bikel. Recovering latent information in treebanks. In *Proceedings of the 19th international conference on Computational linguistics (COLING '02) - Volume 1*, pages 1–7, 2002. 73

[CCCC92] Glenn Carroll, Glenn Carroll, Eugene Charniak, and Eugene Charniak. Two experiments on learning probabilistic dependency grammars from corpora. In *Working Notes of the Workshop Statistically-Based NLP Techniques*, pages 1–13. AAAI, 1992. 32, 35, 37 [cDMMM06] Marie catherine De Marneffe, Bill Maccartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure parses. In *Proceeding of International Conference on Language Resources and Evaluation (LREC 2006)*, 2006. 11

[CGS08] S. B. Cohen, K. Gimpel, and N. A. Smith. Logistic normal priors for unsupervised probabilistic grammar induction. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2008. 6, 35, 37, 85, 94, 99, 100, 101

[Cha00] E. Charniak. A maximum-entropy-inspired parser. In *Proceedings of North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL'00)*, 2000. 5, 8, 11, 32, 57, 68, 71

[Chi99] Zhiyi Chi. Statistical properties of probabilistic context-free grammars. *Comput. Linguist.*, 25:131–160, March 1999. 23

[Cho53] Noam Chomsky. Systems of syntactic analysis. *J. Symb. Log.*, pages 242–256, 1953. 17

[Cho57] N. Chomsky. *Syntactic structures.* Mouton, Den Haag, 1957. 17

[Cho65] Noam Chomsky. *Aspects of the Theory of Syntax.* The MIT Press, 1965. 3, 5

[CM04] Montserrat Civit and Ma Antnia Mart. Building cast3lb: A spanish treebank. *Research on Language and Computation*, pages 549–574, 2004. 95

[CMBN$^+$03] M. Civit, Ma. A. Mart, N. Bufi B. Navarro, B. Fernndez, and R. Marcos. Issues in the syntactic annotation of cast3lb. *4th International Workshop on Linguistically Interpreted Corpora (LINC03) - EACL03*, 2003. 25

[Col96] Michael J. Collins. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 184–191, 1996. 6, 32, 36

[Col97] Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics (EACL '97)*, pages 16–23, 1997. 5, 6, 8, 11, 32, 36, 39, 56, 71, 72, 113

[CS09] S. B. Cohen and N. A. Smith. Shared logistic normal distributions for soft parameter tying in unsupervised grammar induction. In *Proceedings of North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT)*, pages 74–82, 2009. 6, 35, 37, 85, 86, 94, 101

[DDE05] P. Dupont, F. Denis, and Y. Esposito. Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. *Pattern Recognition*, 38(9):1349–1371, September 2005. 98

[DFM04] Michael Daum, Kilian A. Foth, and Wolfgang Menzel. Automatic transformation of phrase treebanks to dependency trees. In *Proceedings of International Conference on Language Resources and Evaluation (LREC 2004)*, pages 1149–1152, 2004. 6

[DIL08] Martín Ariel Domínguez and Gabriel Infante-Lopez. Searching for part of speech tags that improve parsing models. In *Proceedings of the 6th international conference on Advances in Natural Language Processing, GoTAL, Gothenburg, Sweden*, pages 126–137, 2008. 55

[DIL10] Martín Ariel Domínguez and Gabriel Infante-Lopez. Head finders inspection: An unsupervised optimization approach. In *Proceedings of the 7th International conference on Advances in Natural Language Processing, IceTAL, Reykjavik, Iceland.*, pages 127–137, 2010. 71

[DIL11] Martín Ariel Domínguez and Gabriel Infante-López. Unsupervised induction of dependency structures using probabilistic bilexical grammars. In *Proceeding of 7th International Conference on Natural Language Processing and Knowledge Engineering (NLPKE)*, pages 314–318. IEEE, 2011. 85

[DILL10] Martín Ariel Domínguez, Gabriel Infante-López, and Franco Luque. Análisis de dependencias no supervisado basado en gramáticas bilèxicas. In *Proceeding of Workshop on NLP and Web-based technologiesheld in conjunction with IBERAMIA 2010*, 2010. 85

[DLR77] 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*, 39(1):1–38, 1977. 86

[Ear70a] Jay Earley. An efficient context-free parsing algorithm. In *Communications of the ACM*, pages 451–455, 1970. 5

[Ear70b] Jay Earley. An efficient context-free parsing algorithm. *Commun. ACM*, 13:94–102, February 1970. 20

[Eis96] J. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of International Conference on Computational Linguistics (LING-96)*, Copenhagen, 1996. 8, 36, 86

[Eis97] J. Eisner. Bilexical grammars and a cubictime probabilistic parser. In *Proceedings of the International Workshop on Parsing Technologies (IPWT'04)*, pages 54–65, 1997. 8, 36, 56, 72

[Eis00] Jason Eisner. Bilexical grammars and their cubic-time parsing algorithms, 2000. 37

[FK79] W. N. Francis and H. Kucera. Brown corpus manual. Technical report, Department of Linguistics, Brown University, Providence, Rhode Island, US, 1979. 25

[Gai65] Haim Gaifman. Dependency systems and phrase-structure systems. *Information and Control*, pages 304–337, 1965. 34, 35

[GC97] M. Gen and R. Cheng. *Genetic Algorithms and Engineering Design*. John Wiley, 1997. 63

[GGG$^+$10] Jennifer Gillenwater, Kuzman Ganchev, João Graça, Fernando Pereira, and Ben Taskar. Sparsity in dependency grammar induction. In *Proceedings of the ACL 2010 Conference Short Papers*, ACLShort '10, pages 194–199, Morristown, NJ, USA, 2010. Association for Computational Linguistics. 6, 35, 85, 95, 97, 100, 101, 105

[Gol67] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967. 23

[GT07] K. Ganchev Graa and B. Taskar. Expectation maximization and posterior constraints. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2007. 100

[HJM09] W. P. Headden, M. Johnson, and D. McClosky. Improving unsupervised dependency parsing with richer contexts and smoothing. In *Proceedings of North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT)*, pages 101–109, 2009. 6, 8, 35, 37, 85, 86, 92, 94, 99, 101

[Hor69] James Jay Horning. *A study of grammatical inference*. PhD thesis, Stanford University, Stanford, CA, USA, 1969. 25

[HT05] J. Henderson and I. Titov. Data-defined kernels for parse reranking derived from probabilistic models. In *Proceedings of the 43rd Annual Meeting of the ACL*, pages 181–188, 2005. 68

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Publishing, Reading Massachusetts, 1979. 19

[IL05] G. Infante-Lopez. *Two-Level Probabilistic Grammars for Natural Language Parsing*. PhD thesis, Universiteit van Amsterdam, 2005. 67

[ILdR04] G. Infante-Lopez and M. de Rijke. Alternative approaches for generating bodies of grammar rules. In *Proc. 42nd ACL*, 2004. 60, 61

[KL51] S. Kullback and R.A Leibler. On information and sufficiency. In *Annals of Mathematical Statistics*, pages 79–86, 1951. 90

[Kle04] Dan Klein. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting of the ACL*, pages 479–486, 2004. 37

[Kle05] D. Klein. *The Unsupervised Learning of Natural Language Structure*. PhD thesis, Stanford University, 2005. 8, 86, 92

[KM01] D. Klein and C. Manning. Distributional phrase structure induction. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL'01)*, 2001. 57, 72

[KM02] Dan Klein and Christopher D. Manning. Fast exact inference with a factored model for natural language parsing. In *NIPS*, 2002. 39

[KM03a] D. Klein and C. Manning. Accurate unlexicalized parsing. In *ACL*, 2003. 5, 8, 32, 40, 67, 71

[KM03b] Dan Klein and Christopher D. Manning. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems*. MIT Press, 2003. 11

[KM04] D. Klein and C. Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, 2004. 40, 85, 94, 101

[KML03] M.T. Kromann, L. Mikkelsen, and S.K. Lynge. Danish dependency treebank. In *Proceedings TLT*, page pages, 2003. 95

[Mag95a] David M. Magerman. *Natural language parsing as statistical pattern recognition*. PhD thesis, Stanford Univercity, 1995. 32, 71

[Mag95b] David M. Magerman. Statistical decision-tree models for parsing. In *ACL Conference*, 1995. 36, 39, 113

[Mar80] Mitchell Marcus. *A theory of syntactic recognition for natural language*. MIT Press, 1980. 5

[Mar83] I. Marshall. Choice of grammatical word-class without global syntactic analysis: tagging words in the lob corpus. *Computers and the Humanities*, pages 139–150, 1983. 69

[Mar04] Adam Marczyk. Genetic algorithms and evolutionary computation, 2004. 41

[Mel79] Igor A. Mel'čuk. Studies in dependency syntax. *Ann Arbor: Karoma*, pages 23–90, 1979. 4

[MM90] David M. Magerman and Mitchell P. Marcus. Parsing a natural language using mutual information statistics. In *Proceedings of AAAI-90, 8th National Conference on AI*, pages 984–989, 1990. 5, 32

[MMT05] Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Probabilistic cfg with latent annotations. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL '05)*, pages 75–82, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics. 57

[MPRH05] Ryan Mcdonald, Fernando Pereira, Kiril Ribarov, and Jan Haji. 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, 2005. 9

[MR06] M. Mohri and B Roark. Probabilistic context-free grammar induction based on structural zeros. In *Proceedings of North American Chapter of the Association for Computational Linguistics - Human Language Technologies (HLT-NAACL'06)*, 2006. 57

[MS93] M. Marcus and B. Santorini. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19:313–330, 1993. vii, 3, 6, 25, 56, 74, 92

[MS02] C.D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 2002. 15, 16, 28

[NHN05] J. Nilsson, J. Hall, and J. Nivre. MAMBA meets TIGER: Reconstructing a Swedish treebank from antiquity. In *Proceedings of the Nordic Conference on Computational Linguistics (NODALIDA) Special Session on Treebanks*, 2005. 95

[NHN$^+$07] Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. Maltparser: A language-independent system for data-driven dependency parsing. In *Natural Language Engineering*, pages 95–135, 2007. 35, 72

[Niv04] Joakim Nivre. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing (ACL'04)*, pages 50–57, 2004. 37

[Niv05] Joakim Nivre. Dependency grammar and dependency parsing. Technical report, Vxj University, 2005. 15, 33, 34, 37

[NS06] Mark-Jan Nederhof and Giorgio Satta. Estimation of consistent probabilistic context-free grammars. In *HLT-NAACL'06*, pages –1–1, 2006. 23

[Osb00] M. Osborne. Shallow parsing as part-of-speech tagging. In *Conll*, 2000. 69

[OSHTT03] K. Oflazer, B. Say, D. Zeynep Hakkani-Tür, and G. Tür. Building a Turkish treebank. In A. Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*, volume 20 of *Text, Speech and Language Technology*, chapter 15. Kluwer, 2003. 95

[PBE11] Elias Ponvert, Jason Baldridge, and Katrin Erk. Simple unsupervised grammar induction from raw text with cascaded finite state models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 1077–1086, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. 8

[PBK06] S. Petrov, L. Barrett, and D. Klein. Learning accurate, compact, and interpretable tree annotation. In *ACL*, 2006. 57, 70

[Pre01] D. Prescher. Inside-outside estimation meets dynamic EM. In *Proceedings of the 7th International Workshop on Parsing Technologies (IWPT)*, 2001. 86, 100

[Pre03] D. Prescher. A tutorial on the expectation-maximization algorithm including maximum-likelihood estimation and EM training of probabilistic context-free grammars. European Summer School in Logic, Language and Information (ESSLLI), 2003. 41, 47, 48

[PS94] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. The University of Chicago Press, Chicago, 1994. 32

[Ram85] Allan Ramsay. Effective parsing with generalised phrase structure grammar, 1985. 5

[Ris78] J. Rissanen. Modeling By Shortest Data Description. *Automatica*, 1978. 69

[SAJ09] V. I. Spitkovsky, H. Alshawi, and D. Jurafsky. Baby Steps: How "Less is More" in unsupervised dependency parsing. In *NIPS: Grammar Induction, Representation of Language and Language Learning*, 2009. 101

[SAJ10a] V. Spitkovsky, H. Alshawi, and D. Jurafsky. From baby steps to leapfrog: How less is more in unsupervised dependency parsing. In *Proceedings of North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT)*, pages 751–759, 2010. 6, 35, 37, 85, 86, 94, 99

[SAJ10b] Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. Profiting from mark-up: Hyper-text annotations for guided parsing. In *Proceedings of ACL-2010*, 2010. 94, 100

[SAJM10] Valentin I. Spitkovsky, Hiyan Alshawi, Daniel Jurafsky, and Christopher D. Manning. Viterbi training improves unsupervised dependency parsing. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, CoNLL '10, pages 9–17, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. 94, 100

[Sam00] Christer Samuelsson. A statistical theory of dependency syntax, 2000. 37

[SB97] Joan-Andreu Snchez and Jos-Miguel Bened. Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformations, 1997. 23

[Sch72] Roger C. Schank. Conceptual dependency: A theory of natural language understanding. *Cognitive Psychology*, 3(4):pages 532–631, 1972. 4

[SE05] N. Smith and J. Eisner. Guiding unsupervised grammar induction using contrastive estimation. In *IJCAI, Workshop on Grammatical Inference Applications*, pages 73–82, Edinburgh, July 2005. 85, 92

[Seg05] Yoav Seginer. *Learning Syntactic Structure*. PhD thesis, Universiteit van Amsterdam, 2005. 37

[Seg07] Yoav Seginer. Fast Unsupervised Incremental Parsing. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 384–391, Prague, Czech Republic, June 2007. Association for Computational Linguistics. 8

[SJ01] P. Schone and D. Jurafsky. Language-independent induction of part of speech class labels using only language universals. In *IJCAI'01*, 2001. 57

[SOS$^+$02] Kiril Simov, Petya Osenova, Milena Slavcheva, Sia Kolkovska, Elisaveta Balabanova, Dimitar Doikoff, Krassimira Ivanova, Alexander Simov, Er Simov, and Milen Kouylekov. Building a linguistically interpreted corpus of bulgarian: the bultreebank. In *Proceedings of International Conference on Language Resources and Evaluation (LREC 2002), Canary Islands*, page pages, 2002. 95

[SZ09] Federico Sangati and Willem Zuidema. Unsupervised methods for head assignments. In *Proceedings of the eighth conference on European chap- ter of the Association for Computational Linguistics (EACL '09)*, pages 701–709, 2009. 73, 83

[TDdlH00] F. Thollard, P. Dupont, and C. de la Higuera. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *Proceedings of International Conference on Minority Languages (ICML)*, Stanford, 2000. 59, 64, 90

[Tes59] Lucien Tesnière. *Eléments de Syntaxe Structurale*. Klincksieck, 1959. 34

[UK04] Tylman Ule and Sandra Kübler. From constituent structure to dependencies, and back. In *Proceedings of the International Conference on Linguistic Evidence*, 2004. 39

[Whi94] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85, 1994. 41

[WZ10] Zhiguo Wang and Chengqing Zong. Phrase structure parsing with dependency structure. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 1292–1300, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. 6, 35

[You67] Daniel H. Younger. Recognition and parsing of context-free languages in time n^3. In *Information and Control*, pages 189–208, 1967. 5, 20