TOWARDS SIMPLER TREE SUBSTITUTION GRAMMARS

MSc Thesis (*Afstudeerscriptie*)

written by

Federico Sangati
(born December 14th, 1982 in Abano Terme (PD), Italy)

under the supervision of **Dr Willem Zuidema**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:	**Members of the Thesis Committee:**
November 21, 2007	Prof. Dr Peter van Emde Boas
	Prof. Dr Rens Bod
	Dr Willem Zuidema

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

Contents

Acknowledgments

I would like to express my gratitude to all the people who in a way or another helped me throughout this thesis during the last six months.
From a logistic perspective I want to thank the ILLC institute who provided me with a desk and all the people who by consequence had to share more or less literally their working space with me: Gideon Borensztajn, Nina Gierasimczuk, Anouk Perquin, and Fernando Velazquez-Quesada. Not only they had to get used to the idea of a fifth chair in their office, but they also agreed to see and hear their desktop computers "sweating" while heavily working on my experiments in several occasions.
I'm very grateful to Willem Zuidema for supervising my thesis. In all its phases from conceptualization to revision he played a fundamental role. I'm also grateful to Rens Bod for his post-defense comments on the thesis.
Finally, I am thankful to all the people who helped me with partial revision of the thesis and those who contributed with high dosage of moral support, in particular Paipin Cheng, Silvia Gaio, Joeri Honnef and Daniil Umanski.

Amsterdam Federico Sangati
November, 2007.

Chapter 1

Introduction

1.1 Language Learning

In this thesis we will investigate several supervised methods of learning the syntactic structure of natural languages. Supervised learning is one of several machine learning paradigms. It differs from the unsupervised methodologies in the fact that it learns from a number of existing examples of correct syntactic structures.
A parse tree, in “modern” linguistic tradition, is the standard representation of the syntactic structure of a sentence. In a parse tree, we distinguish the tree structure (bracketing) from the labeling of the nodes of the tree. Each label serves to categorize the part of the sentence which lies under its scope. As an example Figure 1.1 shows a simple labeled tree structure.

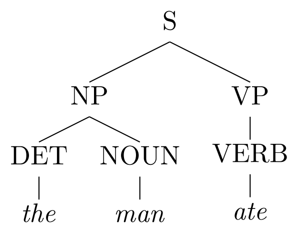


Figure 1.1: A simple parse tree.

There has been controversy about the way to manually produce the syntactic structure of natural language sentences. We will not take part in this discussion, and we will assume that the input annotated corpus correctly represents the linguistic constructions we want to learn. We will consider the manually

annotated syntactic tree structure of the Wall Street Journal as our input tree-bank.
In line with the mainstream computational linguistics tradition, we focus on the syntactic analysis of language. This choice is motivated by the fact that the syntactic structure of a sentence provides the first tool for disambiguating its possible interpretations. Learning how to derive the structure of natural language sentences, is therefore the first step towards language comprehension and production.
The same methodology can also provide an account for the learning process that humans experience during language acquisition. There is in fact no doubt that the utterances a child is exposed to exhibit a certain degree of regularity which is relevant for language acquisition. The research of computational properties of language can therefore shed some light on our scientific understanding of the human language faculty [Abney, 1996].

1.2 Formal Grammars

Formal grammars were made popular by Chomsky [1956], with the intention to give an account for the algebraic aspect of formal languages. In his formulation, each grammar is characterized by a typology of rewriting rules which define a set of grammatical strings over an alphabet of symbols.
Although simple formal grammars are considered, from the theoretical linguistic tradition, not powerful enough to capture the universal properties of human languages, they find direct application in empirical based research, like ours, which is concerned with the task of learning formal grammars from actual language instances.
Each supervised method that we will be dealing with, relies on some notion of formal grammar. In particular we will focus on different ways of defining the basic units of linguistic structures. Each grammar instance that we will consider is therefore related to a way of extracting elementary fragments from input annotated natural language sentences. In general, we will refer to this set of elementary units as the grammar which has been learned.
It is the goal of any learning system to be able to generalize beyond the set of training data. In the linguistic domain this translates to the ability of producing or parsing grammatical sentences not present in the original training corpus. To achieve this we need to define a paradigm to recombine the elementary fragments in our grammar into new syntactic parse trees. In particular, we will be interested in examining the generative ability of the class of tree substitution grammars, which utilizes the substitution operation.

1.3 Probabilistic grammars

1.3.1 Disambiguation problem

The possibility of combining the fragments of our grammar into full syntactic structures, could allow in principle to generate an infinite number of grammatical sentences. However, our goal here is to build a tool for parsing incoming text.
When processing an incoming sentence, we might encounter a large number of possible parse trees, produced by our grammar, which all yield the target sentence. As an example we report the number of possible parses the grammar of Martin et al. [1987] assigns to sentences of increasing lengths.

List the sales of products in 1973	3
List the sales of products produced in 1973	10
List the sales of products in 1973 with the products in 1972	28
List the sales of products produced in 1973 with the products produced in 1972	455

Although the interpretation of these sentences is univocal for English speakers, the grammar of Martin et al. [1987] allows for a number of possible analysis which is exponential in the length of the sentence, most of which are not considered acceptable by language users.
Assuming that there is only one correct interpretation for each sentence, we need to provide our parser with the ability of discriminating between all possible analyses, using some notion of acceptability. The disambiguation task, is then related to the ability of selecting the correct parse out of all the possible ones.

1.3.2 Probabilistic Model

The disambiguation problem can be solved if we supplement our formal grammar with a probabilistic model. In this way we can rank all the different parses of a certain sentence on the base of their probabilities, and distinguish between the interpretations which are grammatically plausible from the possible ones.
In this probabilistic framework, we want to see not only which units take part in the grammar but also how often they occur in the linguistic structures. The intuition behind this is that the frequency of appearance of the units carries valuable information for deciding how to compose them, and will therefore constitute an important part of the learning process. Furthermore, the probability of a parsed tree is estimated by combining the weights of its elementary fragments.

1.3.3 Competence and Performance

The fact of accounting for probabilistic grammars represents a crucial separation from the theoretical linguistic research agenda, which is mainly concerned

with the problem of distinguishing well formed sentences from ungrammatical ones. When dealing with gradients of acceptability, we break the dichotomy between well and ill formed sentences, assumed in language competence models [Chomsky, 1965]. When analyzing a given sentence, the only distinction that we will draw is between the correct parse and all the rest [Charniak, 1997].
An other reason behind our choice of abandoning a strict view on grammaticality is the fact that we are mainly interested on processing actual language production instances, which include possible sloppiness, common mistakes and in general not entirely linguistically kosher sentences. Furthermore, in our methodology, we don't consider language as fixed and immutable but we want to include all those phenomena such as language change and language variation which can only be described within the analysis of the surface structure of the language. The framework of competence models cannot give a proper account for actual language use, language change and language variation and it needs therefore to be replaced by a model of language performance [Scha, 1990].

1.4 Research outline

The rest of the thesis is divided in three main chapters.

In Chapter 2, we start with defining the general class of Tree Substitution Grammars (TSG), and how it is supplemented with a probabilistic model. We will review some of its possible instantiations, namely probabilistic context-free grammars (CFGs), DOP-like grammars, and stochastic Lexicalized Tree Substitution Grammars (LTSGs). These three grammars mainly differ in their domain of locality [Joshi, 2004]: CFGs consider fragments of minimal size, while DOP and LTSGs allow for elementary trees of bigger size. Furthermore LTSGs allow only for lexicalized elementary trees in the grammar.

In Chapter 3, we focus on a sub-class of LTSGs, viz. one-anchored lexicalized tree substitution grammars, characterized by elementary trees that have exactly one lexical item. Similar grammars have been successfully used in models within the more general framework of tree adjoining grammars [Joshi and Schabes, 1997]. In our analysis we will evaluate a number of methods for extracting such grammars from a tree-bank using both standard and heuristic based approaches. The subclass of one-anchored LTSGs under consideration exhibits a good tradeoff between the size of the grammar and the possibility of allowing big syntactic constructions in the set of elementary trees. While the compactness of the grammar makes the model more tractable from a computational perspective, the presence of potentially big syntactic constructions is desirable when we want to account for a certain degree of idiomaticity in the language.
After obtaining quantitative results in terms of performance of different ways of extracting one-anchored lexicalized tree substitution grammars, we have discovered interesting insights. In the first row of results we find two of our own

developed methods to score better than the grammar extracted from the Collins and Magerman head percolation rules. Another interesting finding is that another very accurate grammar is the one resulting from a random assignment of head dependencies. This finding rises relevant questions about the drawbacks of coherent principle-based approaches.

The goal of Chapter 4 is to further analyze the grammars considered in Chapter 3. In particular we will characterize the tree constructions which constitute them and analyze when they fail to generalize over new sentences. Finally we will propose a possible approach that could in principle overcome these failures: the idea is to understand how the less frequent constructions are related to those which occurs more often in the training corpus. We foresee the possibility of defining simple operations to transform general "template" fragments into possible instantiations. The final chapter does not intend to provide an exhaustive analysis of those techniques, and should be therefore considered as a hint for future directions.

Chapter 2

Stochastic Tree Substitution Grammars

2.1 Introduction

The intent of this chapter is to give an overview on different models in the family of the Stochastic Tree Substitution Grammars (STSGs). The various models belonging to this family, mainly differ in the domain of locality [Joshi, 2004] taken under consideration. The domain of locality relates to how much extended the grammatical rules are. The historical baseline is represented by the Context Free Grammar formalism (CFG), which considers only one level trees in the grammatical rules, resulting in a compact and efficient implementation.
The other two classes of models under consideration are Lexicalized Tree Substitution Grammar (LTSG), and Data Oriented Parsing (DOP) [Bod, 1992].
Although members of the same family, these three models present substantial differences in the linguistic assumptions they are based on. The goal of this chapter is to understand the choices behind each model, and how they affect their linguistic accuracy.
Both the LTSG and DOP model exist in many different variations. It is beyond the goal of this chapter to describe them all. We will limit our analysis to some of their instances: we will consider head rules dependencies to extract one of the possible LTSG, and several versions of DOP including DOP1 [Bod, 1998], "Bonnema" estimator [Bonnema and Scha, 2003] and the Goodman reduction [Goodman, 2003].

2.2 STSG

A tree substitution grammar can be represented with a 4-tuple $\langle V_n, V_t, S, T\rangle$ where:

- V_n is the set of nonterminals
- V_t is the set of of terminals
- $S \in V_n$ is the starting symbol (usually "TOP")
- T is the set of elementary trees, having root and internal nodes in V_n and leaf nodes in $V_n \cup V_t$

Trees in T potentially include everything from trees of depth one to trees of indefinite depth. Figure 2.1 shows three instances of elementary trees.

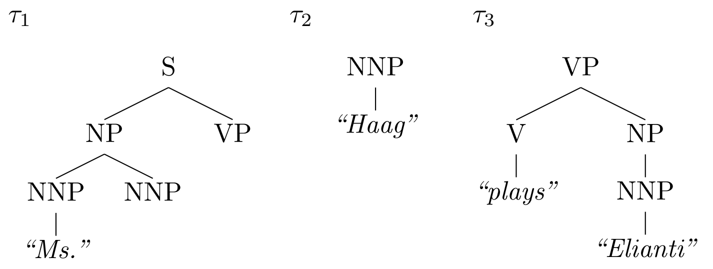


Figure 2.1: Example of elementary trees of maximum depth 3, 1, and 3.

Two elementary trees α and β can be combined by means of the substitution operation $\alpha \circ \beta$ to produce a unified tree, only if the root of β has the same label of the leftmost nonterminal leaf of α. The combined tree corresponds to α with the leftmost nonterminal leaf replaced with β. The substitution operation can be applied iteratively: $\alpha \circ \beta \circ \gamma = (\alpha \circ \beta) \circ \gamma$.
When the tree resulting from a series of substitution operations is a complete parse tree, i.e. all its leaf nodes are terminal nodes, we define the sequence of the elementary trees used in the operations as a derivation of the complete parse tree. Considering the 3 elementary trees introduced before, $\tau_1 \circ \tau_2 \circ \tau_3$ constitutes a possible derivation of the complete parse tree of Figure 2.2.
A stochastic[1] TSG defines a probabilistic space over the set of elementary trees: for every $\tau \in T$, $P(\tau) \in [0,1]$ and $\sum_{\tau':r(\tau')=r(\tau)} P(\tau') = 1$, where $r(\tau)$ returns the root node of τ.

[1]Throughout this thesis, we will be only concerned with probabilistic grammars. The term 'stochastic' will be sometimes omitted for simplicity.

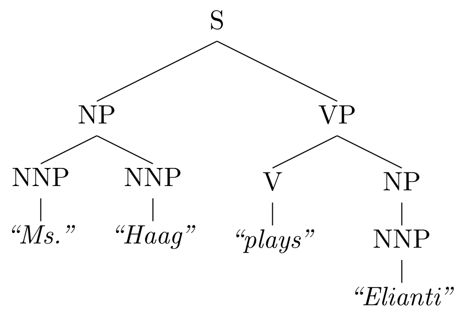
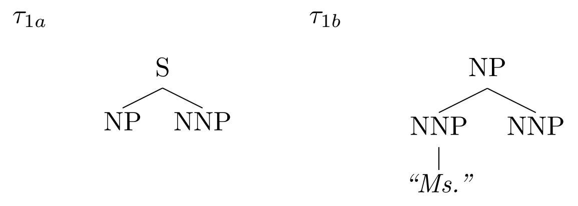


Figure 2.2: Parse tree of the sentence *“Ms. Haag plays Elianti”*.

Having defined a probabilistic space over elementary trees, and a way to combine them, we are now able to assign a probability to every possible derivation. Assuming that the substitution operations used to generate a single derivation are stochastically independent, we define the probability of a derivation as the product of the probability of its elementary trees. If a derivation d is constituted by n elementary trees $\tau_1 \circ \tau_2 \circ \ldots \circ \tau_n$, we will have:

$$P(d) = \prod_{i=1}^{n} P(\tau_i)$$

Depending on the set T of elementary tree, we might have different derivations producing the same parse tree. If for example the tree τ_1 in Figure 2.1 is split in two subtrees, τ_{1a} and τ_{1b}, i.e.



we have a new derivation $\tau_{1a} \circ \tau_{1b} \circ \tau_2 \circ \tau_3$ of the parse tree in Figure 2.2.

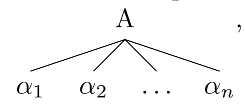
We are now ready to define the probability distribution of a parse trees. For any given parse tree t, we define $\delta(t)$ the set of its derivations generated from the grammar. Since any derivation $d \in \delta(t)$ is a possible way to construct the parse tree, we will compute the probability of a parse tree as the sum of the probabilities of its derivations.

$$P(t) = \sum_{d \in \delta(t)} \prod_{\tau \in d} P(\tau)$$

We have now defined all the general components of the TSG model. The crucial aspect which is left to be defined is the way to extract the elementary trees from a linguistic corpus and how to obtain a probability distribution over these elementary trees. In the following sections we will present three possible approaches: CFG, LTSG, and DOP.

2.3 CFG

Context free grammars represent the simplest implementation of the TSG model just defined, and they will therefore constitute the baseline for our experiments. When considering a corpus of complete parse trees $\mathscr{T}$ the simplest way to extract elementary trees is to consider for each parse tree all possible subtrees of depth 1. More formally a CFG is defined as a 4-tuple $\langle V_n, V_t, S, R\rangle$ where the first 3 elements are identical to the ones defined in the TSG and R represents the set of CFG rules which correspond to elementary trees of depth 1: for every tree of depth one



, we will have a rule in R: A $\rightarrow \alpha_1\ \alpha_2\ \ldots\ \alpha_n$.

Here we have $A \in V_n$ and $\forall j,\ \alpha_j \in V_n \cup V_t$. The root A in rule representation is often define as the left hand side (lhs) of the rule, while the daughters are often called its right hand side (rhs).
Using this rule extraction we can list the rules within the complete parse in Figure 2.2:

$$\begin{aligned}
\text{S} &\rightarrow \text{NP VP}\\
\text{NP} &\rightarrow \text{NNP NNP}\\
\text{VP} &\rightarrow \text{V NP}\\
\text{NNP} &\rightarrow \textit{“Ms.”}\\
\text{NNP} &\rightarrow \textit{“Haag”}\\
\text{V} &\rightarrow \textit{“plays”}\\
\text{NNP} &\rightarrow \textit{“Elianti”}
\end{aligned}$$

In order to define a probabilistic distribution over the CFG rules, we will count for each rule $r \in R$ the occurrence frequency $f(r)$, defined as the number of times rule r was encountered in the training corpus (possibly more than once in the same parse tree):

$$f(r) = \sum_{t \in \mathscr{T}} count(r, t)$$

We are now able to define a probability distribution over the set of rules: we will assign to every $r \in R$ a probability $P(r)$ which equals its relative frequency $F(r)$ which is simply defined as its occurrence frequency divided by the sum of all occurrence frequencies of all the other rules sharing the same left hand side.

$$F(r) = \frac{f(r)}{\displaystyle\sum_{r':lhs(r')=lhs(r)} f(r')}$$

We can easily show that the relative frequency estimator defines a proper distribution on the set of rules, since it satisfies the constraint defined in the TSG model: for each $r \in R$,

$$\begin{aligned}\sum_{r':lhs(r')=lhs(r)} P(r') &= \sum_{r':lhs(r')=lhs(r)} \frac{f(r')}{\displaystyle\sum_{r'':lhs(r'')=lhs(r')} f(r'')} \\ &= \frac{\displaystyle\sum_{r':lhs(r')=lhs(r)} f(r')}{\displaystyle\sum_{r'':lhs(r'')=lhs(r')} f(r'')} = 1\end{aligned}$$

We are now able to define a probability distribution over the complete parse trees in the training corpus. Every observed complete tree can be decomposed in a collection of rules, representing the unique derivation of the tree using the CFG. Assuming that the substitution operations are independent from one another (context-freeness assumption) the probability of the complete parse tree will equal the probability of this unique derivation. More formally, for each $t \in \mathscr{T}$,

$$P(t) = \prod_{r \in t} F(r)^{Count(r,t)}$$

The same formula can be applied to compute the probability of a complete parse tree not observed before, with the only constraint that every rule has to be present in the grammar acquired from the training corpus.
As an ultimate extension of this reasoning, given a probabilistic CFG acquired on a training corpus, and given a flat sentence, we are able to define, in a bottom-up fashion, a parse forest which collects all possible parse trees yielding that sentence. By computing the probabilities of each of the possible parses, we can find the one with the highest probability.

2.4 Lexicalized Tree Substitution Grammars

2.4.1 Introduction

We jave just seen how in every CFG rule, the "domain of locality" is restricted to a single level of the tree. This means that the possibility of applying a rule is assumed to be independent of the global context in which it occurs. For example the rule V $\rightarrow$ *"plays"* producing the predicate in the parse tree in Figure 2.2 doesn't carry any information about the verb's argument. Natural languages are believed to behave between context-free and context-sensitive languages in the Chomsky hierarchy [Joshi, 1985, Shieber, 1985]. For this reason, many criticisms where expressed on the possibility to obtain full linguistic performances with CFG. We might therefore expect better results by looking at richer formalisms.
One way of extending the domain of locality is to allow elementary trees of bigger depth. When considering the problem of extracting elementary trees of arbitrary depth from an input parse tree, one realizes that the number of possible subtrees grows exponentially with the depth of the input tree (we will show this later in section 2.5.4). One way in which we can limit this number is considering only lexicalized subtrees, i.e. trees keeping at least one lexical anchor among its leaf nodes. We will now estimate a lower and upper bound in the number of lexical subtrees of a parse tree with n leaf nodes. Assuming that the parse tree is binary (except for the lexical production) we can come up with the most skewed and the most balanced tree, the first representing the lower bound in the number of lexicalized subtrees and the second the upper bound. Figure 2.3 shows the two extremes when the number of lexical nodes $n = 4$.

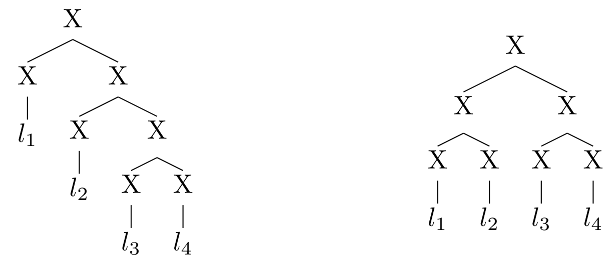


Figure 2.3: Skewed (left) and balanced (right) tree with 4 lexical productions (l_i) with 31 and 34 lexical subtrees respectively. In both cases we have a total number of 27 possible lexical derivations.

Table 2.1 summarizes the calculations to obtain the number of lexical subtrees and derivations in these two kinds of binary trees. The number of lexical derivations (derivations containing only lexical elementary trees) is the same in both cases. In fact if $n = 2^x$, the number of leaf node we have for both the balanced tree and the skewed tree $L\delta = 3^{2^x-1}$. Figure 2.4 shows the exponential growth of the lexical subtrees and derivations in the two extreme cases.
We might want to find other strategies to further constrain the number of lexical elementary trees. One idea is to allow each tree to have a maximum number of lexical anchors with possible limitations on the maximum distance between the left-most and the right-most anchors (if we have two or more anchors per tree). In the following section we will analyze the most restricted case, where each tree has exactly one lexical anchor.

Skewed Tree	Balanced Tree
$h = n$	$h = \log_2 n + 1$
$\sigma_h = 2 \cdot (\sigma_{h-1} + 1)$	$\sigma_h = (\sigma_{h-1} + 1)^2$
$\psi_h = \sigma_h + \psi_{h-1} + 1$	$\psi_h = \sigma_h + 2 \cdot \psi_{n-1}$
$\neg L\psi_h = \neg L\psi_{h-1} + h - 1$	$\neg L\psi_h = \psi_{h-1}$
$L\psi_h = \psi_h - \neg L\psi_h$	$L\psi_h = \psi_h - \neg L\psi_h$
$N_h = 2 \cdot (h - 1)$	$N_h = 2^h - 2$
$P_h = n = h$	$P_h = n = 2^{h-1}$
$L\delta_h = 3^{N_h+1-P_h}$	$L\delta_h = 3^{N_h+1-P_h}$

Table 2.1: Calculation of lexical subtrees and derivations for balanced and skewed binary trees, where h: depth, n: leaf nodes, σ: number of starting subtrees, ψ: number of subtrees, $\neg L\psi$: number of non-lexical subtrees, $L\psi$: number of lexical subtrees, N_h: number of internal nodes, P: number of pre-lexical nodes, $L\delta$: number of lexical derivations

.

2.4.2 Extracting LTSG from head dependencies

In this section we will only be concerned with lexical elementary trees with a single anchor. Moreover we will describe a method to assign to each word token that occurs in the corpus a unique elementary tree. This method depends on the annotation of head-dependent structure in the tree bank, for which we will use the head-percolation rules defined in [Magerman, 1995, Collins, 1997][2]. According to this scheme, each node of every parse tree has exactly one of its children annotated as head. The root of the tree is also annotated as head. As an example we will report in figure 2.5 the parse tree reported before, enriched with head-annotation: the suffix -H indicates that the node is head labeled.

[2]We used the freely available software "Treep" [Chiang and Bikel, 2002] to annotate the treebank with head-dependent structure.

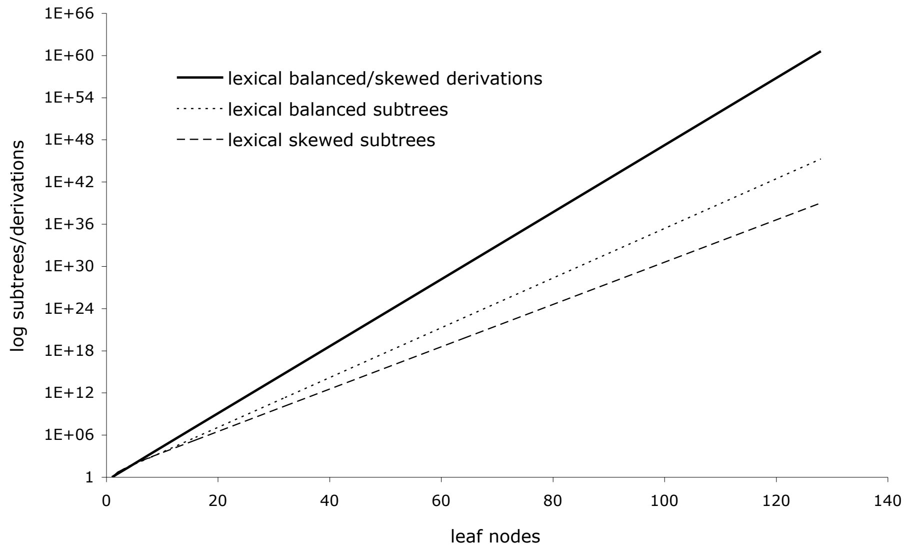


Figure 2.4: Growth of lexical subtrees and derivations for balanced and skewed binary trees of increasing number of leaf nodes.

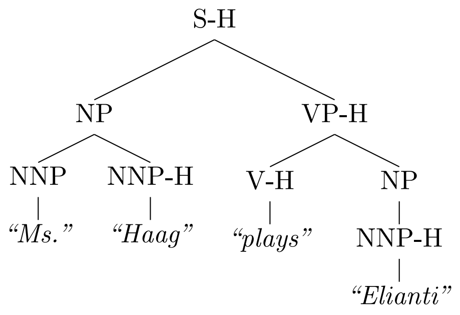
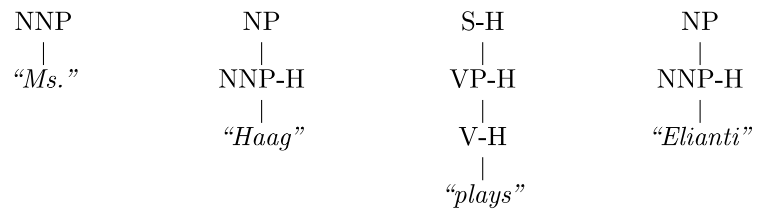


Figure 2.5: Parse tree of the sentence *"Ms. Haag plays Elianti"* annotated with head markers.

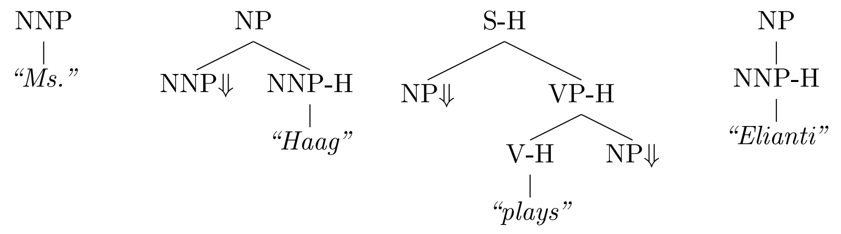
The intuition behind the Collins-Magerman scheme (CM-LTSG) is that structure dependency can be approximately decided from the labels of the internal nodes. For instance if the starting label of a parse sentences is S, the head-percolation scheme will choose to assign the head marker to the first daughter from the left labeled with TO. If no such label is present, it will look for the first IN. If no IN is found, it will look for the first VP, and so on.
Once a parse tree is annotated with head markers in such a manner, we will be able to extract its *lexical paths* (or spines), each having a unique lexical anchor. Starting from each lexical production we need to move upwards towards the root through the head dependencies, until we find the first internal node which is not marked as head (or until we reach the root of the tree). In the example above, the verb of the sentence *"plays"* is connected through head-marked nodes to the root of the tree. In this way we can extract the 4 lexical paths of the parse tree in figure 2.5:



Following this simple procedure we can easily assign to every lexical item a lexical path. What is left to be shown is that, in every case, the extracted lexical paths cover the whole parse tree. Let's assume that this is not the case; there must be an internal node J which doesn't belong to any path. J cannot be a pre-lexical node otherwise it would have been the first percolation step from its lexical item[3]. But if J is not pre-lexical, according to the percolation scheme, it must have a specific daughter which is head marked, and the same applies to this head-marked daughter (if not a pre-lexical node). By following the head dependency path downwards, we will end at one of the lexical items. We have therefore shown that there cannot be any internal node which is excluded from all the extracted lexical paths. Now we want to see if any internal node might occur in more than one path. This is only possible if two percolation paths converge at some point to the same internal node. But this is not possible since every internal node percolates upwards to its mother only if it is head marked and there is only one head marked daughter for each internal non-prelexical node. We have thus shown that every head-annotated parse tree is perfectly covered by the derived lexical paths.

[3]We assume in general that the terminal nodes of a parse tree are unary productions.

The lexical paths which we are now able to extract from any parse tree cannot be combined together because they don't involve substitution sites (non terminal nodes at the frontier of the tree). We should therefore convert every extracted path to an elementary tree, by completing every internal node with the other daughters not present in the path. In this way we have defined a way to obtain a derivation of any parse tree composed of lexical elementary trees. The 4 elementary trees completed from the previous paths are shown below with the substitution sites marked with $\Downarrow$.



2.4.3 Formal LTSGs

More formally LTSGs are defined with the TSG 4-tuple $\langle V_n, V_t, S, T\rangle$ where T is the collection[4] of lexicalized trees extracted from the input treebank. Similarly to the context free rules, for every elementary tree $\tau \in T$ we define the occurrence frequency $f(\tau)$, as the number of times it occurred in the bag of all lexical elementary trees collected from all parse trees in the input corpus. Even in this case, a single elementary tree might occur more than once in the same parse tree. The probability of an elementary tree τ is defined as its relative frequency in the collection of trees sharing the same root label:

$$P(\tau) = \frac{f(\tau)}{\displaystyle\sum_{\tau' : r(\tau') = r(\tau)} f(\tau')}$$

As previously defined in the general case of STSGs, we can define the probability of a parse tree $t \in \mathscr{T}$ as the sum of the probabilities of all its derivations $\delta(t)$. In fact, depending on the the bag T of elementary trees extracted from the training corpus, there might be more than one possible derivation for each parse tree.

[4]In order to capture all the occurrences of all elementary trees we need to store each token we encounter separately. A compact representation of this bag of elementary trees is constructed by storing each tree instance only once, together with a counter keeping track of how many times it occurred in the input data.

$$P(t) = \sum_{d \in \delta(t)} \prod_{\tau \in d} P(\tau)$$

Given an input sentence not present in the training data, we are in principle able to reconstruct all possible parses from the set of lexical elementary trees yielding that sentence, and choose the one with maximum probability. In reality the set of elementary trees cannot always produce a parse tree for every test sentence. To increase the chance to reconstruct a parse tree, we separate the lexical productions of every elementary tree in order to create new lexical trees, as reported in other models [Chiang, 2003]. In this way our set T of elementary tree will be constituted by delexicalized trees and production rules. The new grammar, more compact and with more generalization power, has a bigger chance to produce a parse tree for a test sentence.

2.5 DOP

2.5.1 Introduction

Data Oriented Parsing is the third and last model we will take into consideration in this chapter. This approach differs from the the majority of the language processing models (including the previous two) in the assumptions it makes on the representation of linguistic information. Without denying the existence of grammatical rules at the base of natural language structure, it considers equally important the memorization of the combination of linguistic fragments from the linguistic experience. This is in accordance with the hypothesis that languages exhibit a certain degree of idiomaticity which is not fully explainable from a purely rule based grammar.

2.5.2 DOP1

We will here restate the basic probability model behind the first implementation of DOP [Bod, 1993]. As with LTSG, we will not constrain the depth of the elementary trees extracted from the input corpus, but unlike LTSG we will not require elementary trees to be lexicalized. Specifically this approach considers all possible subtrees of every input parse tree as a valid elementary tree (each subtree belonging to a specific position of an input parse tree is extracted once).

Given an elementary tree $\tau \in T$ its occurrence frequency $f(\tau)$ is again defined as the number of times it occurred in the bag of subtrees collected from all parse trees in the input corpus. The probability of an elementary tree τ is defined as in the LTSG as its relative frequency $F(\tau)$ which equals its occurrence frequency divided by the sum of all occurrence frequencies of all the elementary trees sharing the same root node with τ.

As previously defined in the general case of STSGs and LTSGs, the probability of every parse tree t is defined as the sum of the probabilities of all its derivations $\delta(t)$ generated from the bag T of subtrees.

$$P(t) = \sum_{d \in \delta(t)} \prod_{\tau \in d} P(\tau)$$

If we want to assign a probability to a certain parse tree t in our treebank $\mathscr{T}$, we must take into consideration all its possible derivations, which are the ones generated from the set of its subtrees. We enumerate all possible $|\delta(t)|$ derivations $d_1, d_2, \ldots, d_{|\delta(t)|}$. Every derivation d_i can be represented as a set of m_j elementary trees $\tau_{i,1}, \tau_{i,2}, \ldots, \tau_{i,m_j}$, such that $t = \tau_{i,1} \circ \tau_{i,2} \circ \ldots \circ \tau_{i,m_j}$, where $\circ$ is the substitution operation previously defined. The probability of the parse tree t then becomes:

$$P(t) = \sum_{i=1}^{|\delta(t)|} \prod_{j=1}^{m_j} P(\tau_{i,j})$$

One of these possible $|\delta(t)|$ derivations is the one generated by the CFG model (the one where all the elementary trees have depth 1), and another one is the one generated by the LTSG if the parse tree would be annotated with head dependencies.
In a parse tree t of $N(t)$ internal nodes (non root and non leaf), these are two of the possible $2^{N(t)}$ derivations. Every derivation is in fact completely defined when we decide which internal nodes are substitution sites.

2.5.3 Parsing with DOP1

If we want to assign a parse tree to an unseen test sentence, we would need to consider all possible parse trees compatible with our grammar yielding that sentence, and within every parse tree summing over all possible derivations originated from the collection T of elementary trees (as defined in the general TSG model).
We first have to notice that there might be no parse tree compatible with our grammar. This case occurs only when a CFG trained on the same input corpus, is not able to generate any parse tree (usually when one or more words in the input sentence are encountered for the first time).
Assuming there is exactly one compatible parse tree for a test sentence, we should understand that not all its subtrees are necessarily present in the collection T of elementary trees. As a consequence not all its possible derivations are necessarily generated from T.
In general, if we have an input sentence not present in the training corpus, and if we consider all its parse trees generated from our grammar, none will be fully represented in all its derivations by the set T of elementary trees. The probability of a parse tree of a sentence, is always smaller than if that parse

tree would be present in our training corpus. This problem[5] is hardly avoidable and results from the fact that every training parse tree is considered to be fully representative of the natural language under consideration.
If the training corpus is sufficiently large, we usually have a representative subset of possible parse trees for every new input sentence, each with potentially many derivations. Unfortunately the problem of computing the most probable parse is NP-complete [Sima'an, 1996], with the bottleneck being the search space over the set of possible derivations, increasing exponentially with the length of the input sentence.
One possible solution to overcome this problem is to reduce the size of T. One way to do this is to keep all subtrees of depth 1 and a fixed number of subtrees of bigger depth up to a certain limit h [Bod, 2001]. We will describe this approach (DOP$_h$) in section 2.6.3. An other strategy consists in approximating the most probable parse tree with the n-most probable derivations (merging the probability of the ones belonging to the same parse tree). In section 2.5.6 will follow this direction using the efficient Goodman reduction algorithm [Goodman, 2003].

2.5.4 Problems with DOP1

In DOP1 the probability of applying an elementary tree in a derivation, equals its relative frequency in the collection of all elementary trees with the same root node.
Considering all possible elementary trees sharing the same root, we know that the sum of their relative frequency is trivially 1. What is not trivial is to understand how this probability mass is divided among the parse trees they generate. We have to remember that the bag of elementary trees is collected from all possible subtrees of each parse tree. In particular given a tree of depth h (not necessarily rooted on TOP) we consider all its possible starting subtree of depth $1, 2, \ldots, h$. We define a tree t_a to be a starting subtree of a tree t_b if t_a is a subtree of t_b and if both share the same root node (they both *start with* the same node).
We will now show that the number σ_h of starting subtrees of a given tree of depth h grows exponentially with the increase of the depth h. If the initial tree is balanced and binary, σ_1 is trivially 1 and $\sigma_i = (\sigma_{i-1} + 1)^2$. In fact any such starting subtree has on the left branch one of the possible σ_{i-1} subtrees *starting with* the left daughter or the trivial subtree constituted by the left daughter alone, and similarly $\sigma_{i-1} + 1$ possible subtrees *starting with* the right daughter on the right branch.
We have for instance that $\sigma_2 = 4$, $\sigma_3 = 25$, $\sigma_4 = 676$, $\sigma_5 = 458329$, σ_6 $2.1 \cdot 10^{11}$ and so on. As shown in Figure 2.6, this exponential growth entails that $\lim\limits_{h\to\infty} \dfrac{\sigma_h}{\sigma_1 + \ldots + \sigma_h} = 1$ if the parse tree is balanced (1/2 if it is skewed).

[5] CFGs don't suffer from this problem while LTSG do (for a smaller degree).

The fraction $\sigma_h / \sum_{i=1}^{h} \sigma_i$ gives the relative frequency of the biggest subtrees of a certain category. The fact that this fraction approaches 1 (1/2 for skewed trees), has undesirable consequences on how the probability mass is distributed. In fact large parse trees make a disproportionately large contribution to the probability mass of the fragments [Bonnema et al., 1999].
To understand things better we provide a simple example. Let's consider a treebank with only two distinct parse trees t_1 and t_2, differing in depth ($t_1 \gg t_2$) and sharing the same root TOP. When extracting all the elementary trees from the two trees, and consider only the fragments rooted on TOP, we will notice that almost all the relative frequency goes to the biggest starting subtrees of t_1.

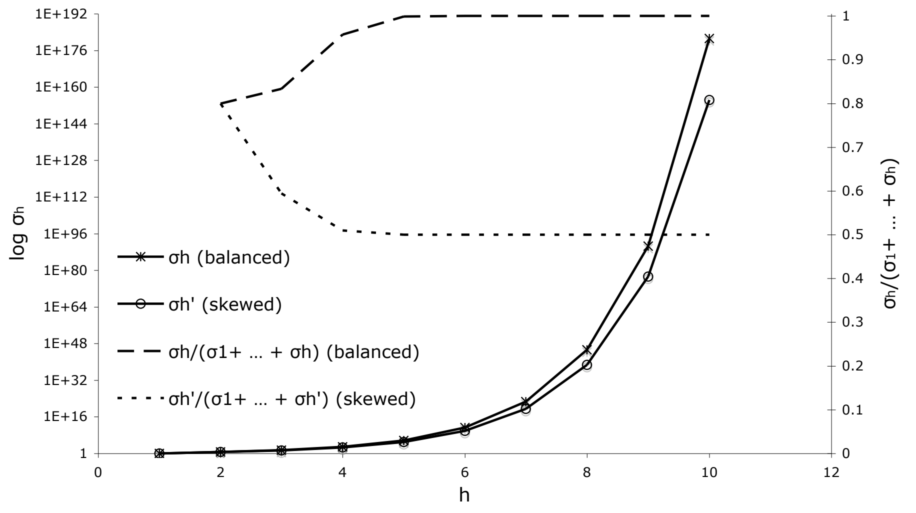


Figure 2.6: Number of initial subtrees σ_h of balanced and skewed parse tree of depth h and h' respectively (straight lines, left axis) and $\sigma_h / \sum_{i=1}^{h} \sigma_i$ (dashed lines, right axis). Each data point related to the skewed parse trees is compared to the one of the balanced tree with the same number of leaf nodes: $h' = 2^{h-1}$.

2.5.5 New estimators for DOP

A way to distribute more fairly the frequency mass is to normalize the frequency of each fragments according to the portions of derivations in which it occurs. We will show this point more formally. Following Bonnema and Scha [2003], given an input parse tree t, we can calculate precisely in how many derivation a certain subtree τ occurs. If $N(t)$ and $N(\tau)$ are the number of non-root non-terminals node of t and τ respectively, we have that the total number of derivation of t, $\delta(t) = 2^{N(t)}$, as explained before, and the number of derivations where τ occurs

is $\delta(t,\tau) = 2^{N(t)-N(\tau)}$ if τ is a starting subtree of t, $\delta(t,\tau) = 2^{N(t)-N(\tau)-1}$ otherwise. In fact each derivation of t where τ occurs is obtained by deciding which of the $N(t)$ non terminals of t are possible substitution sites excepts the ones occupied by the $N(\tau)$ non terminals of τ. If τ is not an initial tree, the position of its root should always be a substitution site.
It follows that the fraction $\mathscr{D}(t,\tau)$ of all derivations of t where τ occurs is:

$$\mathscr{D}(t,\tau) = \frac{\delta(t,\tau)}{\delta(t)} = \begin{cases} 2^{-N(\tau)} & \text{if } \tau \text{ is a starting subtree of } t \\ 2^{-N(\tau)-1} & \text{otherwise} \end{cases}$$

We can come up with an alternative estimator which normalizes the frequency of each elementary tree, by normalizing it with the portion of derivations in which it occurs[6]. The new expected frequency f_N is therefore defined as:

$$f_N(\tau) = \sum_{t \in \mathscr{T}} count(\tau, t)\mathscr{D}(t,\tau)$$

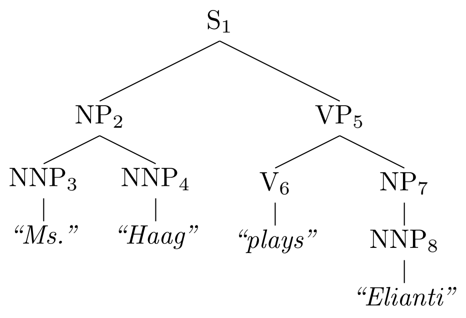
If we assume that in all parse trees τ is either always a starting tree or always a non starting tree we have that $\mathscr{D}(t,\tau)$ is independent from t and therefore:

$$f_N(\tau) = \begin{cases} 2^{-N(\tau)}f(\tau) & \text{if } \tau \text{ is a starting subtree} \\ 2^{-N(\tau)-1}f(\tau) & \text{otherwise} \end{cases}$$

2.5.6 The Goodman reduction

Goodman [1996] was able to define a way to convert the DOP grammar in a novel CFG, of which the size increases linearly in the size of the training data. The perspective which this new implementation is taking is to have a CFG generating every elementary tree with the correct probability. Specifically if we assume that every elementary tree occurs exactly once in our collection T, and we indicate with a the number of elementary trees of category A, we want that the probability of every elementary tree of category A should equal $1/a$.
In order to have every elementary tree occurring exactly once in T we would need to assign to every non-terminal node of every input parse tree a unique index. The original non-terminals are called “exterior” while the ones receiving a unique index are called “interior”. The sentence of Figure 2.2 now becomes:

[6]This is slightly different from the estimator reported in [Bonnema and Scha, 2003], the difference being that they incorrectly normalize the relative frequencies while we normalize the occurrence frequencies.



If we extract all possible elementary trees from an input parse tree labeled in such way, and decompose them in CFG rules, we would "loose" the trace of the original elementary trees. In order to keep track of the boundaries of each elementary tree we should differentiate the root and the substitution sites from the internal nodes; we do so by keeping them in the exterior representation (we say that the elementary tree is in Goodman form). Figure 2.7 shows an example of how an elementary tree is represented in Goodman form.

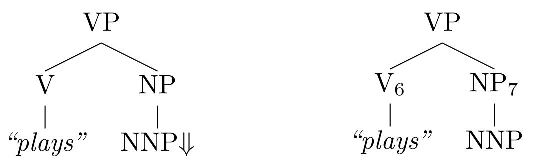
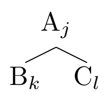


Figure 2.7: TSG elementary tree (left) and the way it is represented in Goodman form.

It's important to notice that even in such representation each elementary tree cannot be extracted more than once from the input treebank. Each elementary tree rooted in A should still have a probability $1/a$. If we indicate with a_j the number of non-trivial subtrees of each parse tree headed by the node A_j we have that $a = \sum_j a_j$.

Now we will explain how to generate a set of PCFG rules from our input treebank, which will generate every elementary tree in Goodman form with the correct probability. Assuming that all input parse trees have only unary or binary productions, for every binary construction



we will generate the following PCFG rules with the corresponding probabilities in parentheses :

$$\begin{array}{llll}
\mathrm{A}_j \rightarrow \mathrm{BC} & (1/a_j) & \mathrm{A} \rightarrow \mathrm{BC} & (1/a) \\
\mathrm{A}_j \rightarrow \mathrm{B}_k\mathrm{C} & (b_k/a_j) & \mathrm{A} \rightarrow \mathrm{B}_k\mathrm{C} & (b_k/a) \\
\mathrm{A}_j \rightarrow \mathrm{BC}_l & (c_l/a_j) & \mathrm{A} \rightarrow \mathrm{BC}_l & (c_l/a) \\
\mathrm{A}_j \rightarrow \mathrm{B}_k\mathrm{C}_l & (b_kc_l/a_j) & \mathrm{A} \rightarrow \mathrm{B}_k\mathrm{C}_l & (b_kc_l/a)
\end{array}$$

The rules in the left side belong to elementary trees where A$_j$ is an internal node, while the right rules belong to elementary trees where A$_j$ is the root node[7] (we will call the first ones "internal" rules, and the second ones "initial" rules). In both cases, each of the two non terminal daughters of A$_j$ can either be a substitution site or not, leading to 4 possible combinations.
While the internal rules and the lower three initial rules are specific to a unique parse tree, the first initial rule (A $\rightarrow$ BC) might appear in multiple constructions (of possibly different parse trees) having in common the same starting category A. If the same rule is encountered more than once, its probability is the sum of the probability of each single instance.

If A$_j$ has only one daughter, the non-terminal B$_k$, we will need to generate the following PCFG:

$$\begin{array}{llll}
\mathrm{A}_j \rightarrow \mathrm{B} & (1/a_j) & \mathrm{A} \rightarrow \mathrm{B} & (1/a) \\
\mathrm{A}_j \rightarrow \mathrm{B}_k & (b_k/a_j) & \mathrm{A} \rightarrow \mathrm{B}_k\mathrm{C} & (b_k/a)
\end{array}$$

If A$_j$ has only one daughter and it is a lexical terminal l we will only have to generate the first rule for each side:

$$\mathrm{A}_j \rightarrow l \quad (1) \quad \mathrm{A} \rightarrow l \quad (1/a)$$

This PCFG produces a proper probability model: the total sum of the probability of the rules on either side sum up to 1. We show this for the binary case (it works analogously for the unary productions):

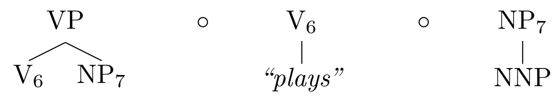
$$\sum_{X \in rhs(A_j)} P(A_j \rightarrow X) = \frac{1 + b_k + c_l + b_kc_l}{a_j} = \frac{1 + b_k + c_l + b_kc_l}{(b_k+1)(c_l+1)} = 1$$

$$\sum_{X \in rhs(A)} P(A \rightarrow X) = \frac{1}{a}\sum_j (1 + b_{k_j} + c_{l_j} + b_{k_j}c_{l_j}) = \frac{1}{a}\sum_j a_j = 1$$

What is left to be shown is that every elementary tree (in Goodman form) headed by the node A, is generated with this PCFG with probability $1/a$. Every

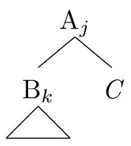
[7]If A$_j$ is the root of a parse tree, the rules in the left side will never be used and they can therefore be omitted.

Goodman form elementary tree is generated by a CFG derivation $\alpha_1 \circ \alpha_2 \circ \ldots \circ \alpha_n$ where each α_i is a 1 level subtree corresponding to a CFG rule, and where only α_1 is an initial rule. For instance the elementary tree in Goodman form, represented in Figure 2.7, will be generated by the following CFG derivation:

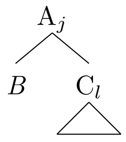


We will for now take into consideration only CFG derivations where all the α_i are internal rules, leading to trees with external non-terminals at the leaf nodes. Each of these derivation will generate a final part of an elementary tree. The probability assigned to each rule can be understood, if we remember the way we calculate the upper bound on the number of subtrees of a binary tree. If a_j is the number of subtrees headed by A_j, and A_j has two daughters, B_k and C_k, each of them could either be a substitution site or not, leading to $(b_k+1)(c_l+1)$ possible subtrees. Each of the internal rules gets a fraction of this total amount of subtree. For instance the rule $A_j \rightarrow B_k C$, gets a probability of b_k/a_j, since there are b_k subtrees headed by B_k that can be substituted in the left daughters and no internal rules can be substituted in the right one (C is a substitution site).

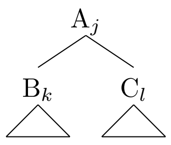
From this reasoning we can conclude that every such derivation leading to a tree headed by A_j gets a probability of $1/a_j$. The formal proof is given by induction on the size of the tree. For the one level deep tree $A_j \rightarrow BC$ the probability is trivially $1/a_j$. Assuming that this holds for any tree headed by B_k or C_l having maximum depth n, using all the rules in the left side we have three possible way to generate a tree of maximum depth $n+1$. In all cases the correct probability is obtained:



$$\frac{b_k}{a_j} \cdot \frac{1}{b_k} = \frac{1}{a_j}$$



$$\frac{c_l}{a_j} \cdot \frac{1}{c_l} = \frac{1}{a_j}$$



$$\frac{b_k c_l}{a_j} \cdot \frac{1}{b_k} \cdot \frac{1}{c_l} = \frac{1}{a_j}$$

The tree generated by any such derivation, can now become an elementary tree (in Goodman form) if substituted in some initial rule. In this case the total derivation of a tree headed by the external non terminal A will have a probability $1/a$. The proof is identical to the one in the previous case, by replacing A_j with A and a_j with a. We have thus shown that this PCFG generates elementary tree in Goodman form with the correct probability.
Besides the standard version of the Goodman reduction explained in this section, there are a number of different implementations taking into consideration "Bonnema-like estimators" [Goodman, 2003].

2.6 Implementation details

2.6.1 Parsing CFGs

In order to parse CFGs we will use BitPar [Schmid, 2004], a freely available parser. It is based on an efficient implementation of the CYK algorithm [Kasami and Torii, 1969] which is used to generate a compact representation of the parse forest for every input test sentence. The same application dynamically calculates the best parse using the Viterbi algorithm [Viterbi, 1967].
In order to deal with unknown words occurring in the test set, we supply BitPar with an open-tag-file which contains the most frequent POS-tags of the words occurring once in the grammar, together with their relative frequency. If an incoming sentence presents an unknown word, BitPar will try to assign to it all possible POS-tags in the open-tag-file, in order to generate the parse forest of the sentence.
The table which the CYK algorithm uses to efficiently store the parse forest requires that the rules in the grammar are in Chomsky-Normal-Form (CNF): every elementary tree corresponding to each rule needs to be binarized; if the number n of daughters (rhs) is greater than 2 we need to substitute it with $n-1$ rules in CNF form. With the reverse procedure we are able to transform the binary rules back to their original format.

non CNF	CNF
$A \rightarrow \alpha_1\ \alpha_2\ \ldots\ \alpha_n$	$A \rightarrow \alpha_1\ A_{\sim 1}$
	$A_{\sim 1} \rightarrow \alpha_2\ A_{\sim 2}$
	$\ldots$
	$A_{\sim n-2} \rightarrow \alpha_{n-1}\ \alpha_n$

2.6.2 Using CFGs for TSG parsing

We will briefly describe how to set up a TSG parser using the CFG formalism. In this way we can use BitPar for both LTSGs and DOP grammars.
In the training phase, according to the specific TSG under consideration, every input parse tree is decomposed into elementary trees, transformed to CNF and delexicalized. When all the trees in the input data are processed and all the elementary trees collected in the bag of elementary trees T, we are able to

calculate the relative frequency of each tree.
Every elementary tree should be now considered by our parser as a unique block which cannot be further decomposed. But if we want to deal with it from a CFG perspective, we eventually need to break it into trees of depth 1. In order to keep the integrity of every elementary tree we will assign to its internal node a unique label. We will achieve this adding "@i" to each i-th internal node encountered in T.
We then read off a PCFG from the elementary trees taking into consideration the original frequencies. In this way the PCFG is equivalent to the original STSG: it will produce exactly the same treess with the same probabilities.
We can now use BitPar to generate the parse forest of an incoming test sentence. Specifically, we will obtain the n-best parses for every test sentence through the Viterbi algorithm. This will allow for possibly more than one derivation for some of the parse trees of the sentence. Finally, in order to compute the most probable parse, we will need to remove the unique labels from the n-best parses, back transform them from CNF and sum the probabilities of the parses which turn out to be identical (each originated from a different derivation). We can now take the parse tree with the highest probability as the best parse of the input sentence.

2.6.3 Implementing DOP$_h$

The exponential growth of the number of derivations in the DOP1 model makes this approach infeasible from a computational point of view. One way of approximating the original approach to a more tractable implementation is by limiting the number of elementary trees. In particular we will consider elementary trees up to depth h. In Algorithm 1 we will report the exact procedure behind the DOP model constrained on subtrees up to a depth h (DOP$_h$). Specifically we will keep all elementary trees of depth 1 and a certain number M of elementary trees of depth $2, \ldots, h$.
In this procedure we need to select a certain number of subtrees of depth $2, 3, \ldots, h$. Bod [2001] chooses to sample 400,000 elementary trees for each depth randomly generated upon their ramification structure, up to depth 14. In our implementation we will select a maximum of $M = 50{,}000$ elementary trees of depth bigger than 1 chosen according to their probability.

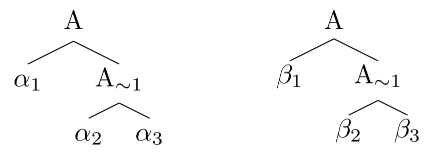
2.6.4 A note on CNF-conversions

It's important to mention that when considering subtrees up to a certain depth we need to use a different version (CNF†) of the standard CNF conversion defined in Section 2.6.1. The standard one could in fact create some problems: if a certain nonterminal A has more than 2 daughters, $A \rightarrow \alpha_1\ \alpha_2\ \ldots\ \alpha_n$, after binarization some of its subtrees (the ones of depth $1, 2, \ldots, h$) result in a subpart of the rule; in this way we fail to have a precise correspondence between the subtrees generated by the original parse tree and the ones generated by the binarized version. As a consequence our grammar would allow to substitute

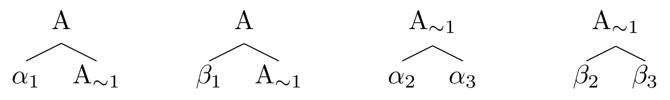
Algorithm 1 DOP$_h$

```
for all parse tree $t \in \mathscr{T}$ do
    transform $t$ to CNF$^\dagger$
    put all subtrees of $t$ of depth 1 in $T$
    for $i = 2$ to $h$ do
        put $M/(h-1)$ subtrees of $t$ of depth $i$ in $T$
    end for
end for
for all elementary tree $\tau \in T$ do
    add unique internal label
    extract CFG rules
end for
```

some of the truncated subtrees in other non compatible elementary trees.
To make things a bit clearer we come up with an example. Let's consider a nonterminal A which occurs in 2 different parse trees in both case with 3 daughters ($A \rightarrow \alpha_1\alpha_2\alpha_3$, $A \rightarrow \beta_1\beta_2\beta_3$). The standard CNF transformation leads to this two trees.



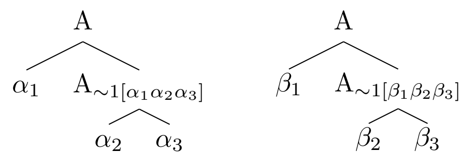
When we extract all possible subtrees up to depth h, we encounter, among others, these four subtrees:



We can notice that the third subtree is substitutable in the second one, and the fourth in the first. These substitutions should not be allowed since the subtrees they combined belong to different rules and could never be combined in the non binarized version of the original parse trees. We therefore need to make sure that the whole rule remains substitutable only within equivalent rules, i.e. rules sharing the same *rhs*. To achieve this we will binarize our rule in the following way:

non CNF	CNF[†]
	$A \rightarrow \alpha_1 \; A_{\sim 1[\alpha_1 \alpha_2 \ldots \alpha_n]}$
$A \rightarrow \alpha_1 \; \ldots \; \alpha_n$	$A_{\sim 1[\alpha_1 \alpha_2 \ldots \alpha_n]} \rightarrow \alpha_2 \; A_{\sim 2[\alpha_1 \alpha_2 \ldots \alpha_n]}$
	$\ldots$
	$A_{\sim n-2[\alpha_1 \alpha_2 \ldots \alpha_n]} \rightarrow \alpha_{n-1} \; \alpha_n$

In the example above the two rules will now present unique identifiers in each internal node, which will prevent from substituting subtrees in wrong places.


A
α_1
$A_{\sim 1[\alpha_1 \alpha_2 \alpha_3]}$
α_2
α_3
A
β_1
$A_{\sim 1[\beta_1 \beta_2 \beta_3]}$
β_2
β_3

2.7 Model Evaluation

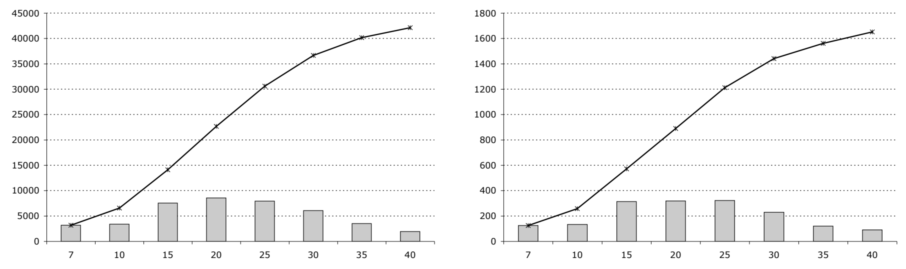
2.7.1 Linguistic Corpus

The evaluation of the different probabilistic models are carried out on the Wall Street Journal corpus [Marcus et al., 1994]. Section 00-21[8] (~42,000 sentences) are used as training data, available in the form of a treebank that contains already parsed sentences. Sentences of section 22 (~1700 sentences) are used as the test set. In order to better compare the different models, the training procedure will be conducted for different lengths of the input sentences (7, 10, 15, 20, 25, 30, 35, 40), whose distribution is shown in Figure 2.8 both for the training and the testing set. To calculate the length of a sentence we ignore traces and punctuation symbols (all words labeled with [-NONE- , : " " .]). Each test sentence is given to the model under evaluation as a flat (non-parsed) sentence, and the best computed parse (test) is compared with the correct one (gold). Labeled Recall (LR) and Precision (LP) together with their harmonic mean (F1) are calculated as evaluation metrics[9]. If C_t and C_g are the set of constituents in the test and gold set respectively, we define:

$$LR = \frac{|C_t \cap C_g|}{|C_g|} \qquad LP = \frac{|C_t \cap C_g|}{|C_t|} \qquad F1 = \frac{2 \cdot LR \cdot LP}{LR + LP}$$

[8]In the literature the usual training is done on sections 02-21. The inclusion in our training of the first two test sections is due to an initial mistake. We decided to continue using the same training set for consistency reason. We believe that results with the correct sections compared with the current ones could help understanding how less training date would affect the different methodologies.

[9]Precision and recall are calculated using EVALB [Sekine and Collins]. We assigned a standard tree for the sentences which could not be parsed: every word is considered a constituent of category NNP.



(a) Sections 0-21 (b) Section 22

Figure 2.8: Distribution of sentences of different lengths in the WSJ. The solid curve represents the cumulative distribution.

2.7.2 Results

We will report the results of the main 3 models described in this chapter (CFG, CM-LTSG, DOP). CM-LTSG is an abbreviation referring to the LTSG extracted from Collins and Magerman head-percolation rules. DOP was implemented in several versions (DOP-Goodman, DOP$_{h2}$, DOP$_{h3}$, DOP$_{h2}$-Bonnema) although for most of them the algorithm was only possible to be trained on sentences of maximum length 7.

We will first compare the size of the grammar generated by the different models for increasing lengths of the sentences in the training corpus. As evaluation metrics of the grammar size we consider both the number of CFG rules and the number of elementary trees (for CFG and DOP-Goodman this two metrics coincide).

Figure 2.9 shows the size of the grammars in the different models using these two metrics, together with the normalization versions, which take into consideration the distribution of the training corpus in Figure 2.8.

Table 2.2 reports the labeled precision, recall and F1 score of the three models. The CFG model could be evaluated using the entire test set, CM-LTSG with sentences up to length 25 and DOP with sentences up to length 10. The F1 results of the two table are visualized in Figure 2.10. Finally Table 2.3 compares the performance of different versions of the DOP model[10].

[10]In the implementation of these models we had access to machines with up to 2GB of memory.

	CFG			CM-LTSG			DOP		
	LR	**LP**	**F1**	**LR**	**LP**	**F1**	**LR**	**LP**	**F1**
7	47.95	53.11	50.39	44.26	31.56	36.85	54.64	50.87	52.69
10	60.85	55.38	57.99	57.64	56.53	57.08	63.35	64.98	64.15
15	65.39	61.25	63.25	69.06	70.32	69.69			
20	66.68	62.93	64.75	73.54	74.11	73.82			
25	67.15	63.94	65.51	75.56	75.90	75.73			
30	67.34	63.97	65.61						
35	64.72	67.40	66.03						
40	64.84	67.60	66.19						
7	85.25	87.12	86.18	84.91	84.45	84.68	88.20	90.14	89.16
10	84.79	81.33	83.02	83.11	82.35	82.73	88.47	89.75	89.11
15	79.94	76.99	78.44	82.66	83.22	82.94			
20	76.50	72.73	74.57	81.54	82.14	81.84			
25	72.80	69.65	71.19	80.93	80.53	80.73			
30	70.41	67.02	68.67						
35	66.45	69.18	67.79						
40	65.65	68.47	67.03						

Table 2.2: Comparison of the results of CFG, CM-LTSG, and DOP trained on WSJ sec 0-21, using increasing sentence length $(7, 10, \ldots, 40)$. Labeled precision, recall and F1 score refer to sentences in section 22 of length less than 60 words (above) and less than the respective maximum length of the training sentences (below).

	WSJ-60			WSJ-7		
	LR	**LP**	**F1**	**LR**	**LP**	**F1**
DOP-Goodman	54.64	50.87	52.69	86.86	86.63	86.74
DOP$_{h2}$	53.42	54.90	54.15	88.20	90.14	89.16
DOP$_{h2}$ **(delex.)**	51.78	54.29	53.01	86.33	88.46	87.38
DOP$_{h3}$	51.28	53.25	52.25	86.60	88.01	87.30
DOP$_{h2}$ **(Bonnema)**	51.14	53.79	52.43	86.60	88.49	87.53

Table 2.3: Comparison of different versions of DOP trained on WSJ-7. Labeled precision, recall and F1 score refer to sentences in section 22 of length less than 60 and 7 words.

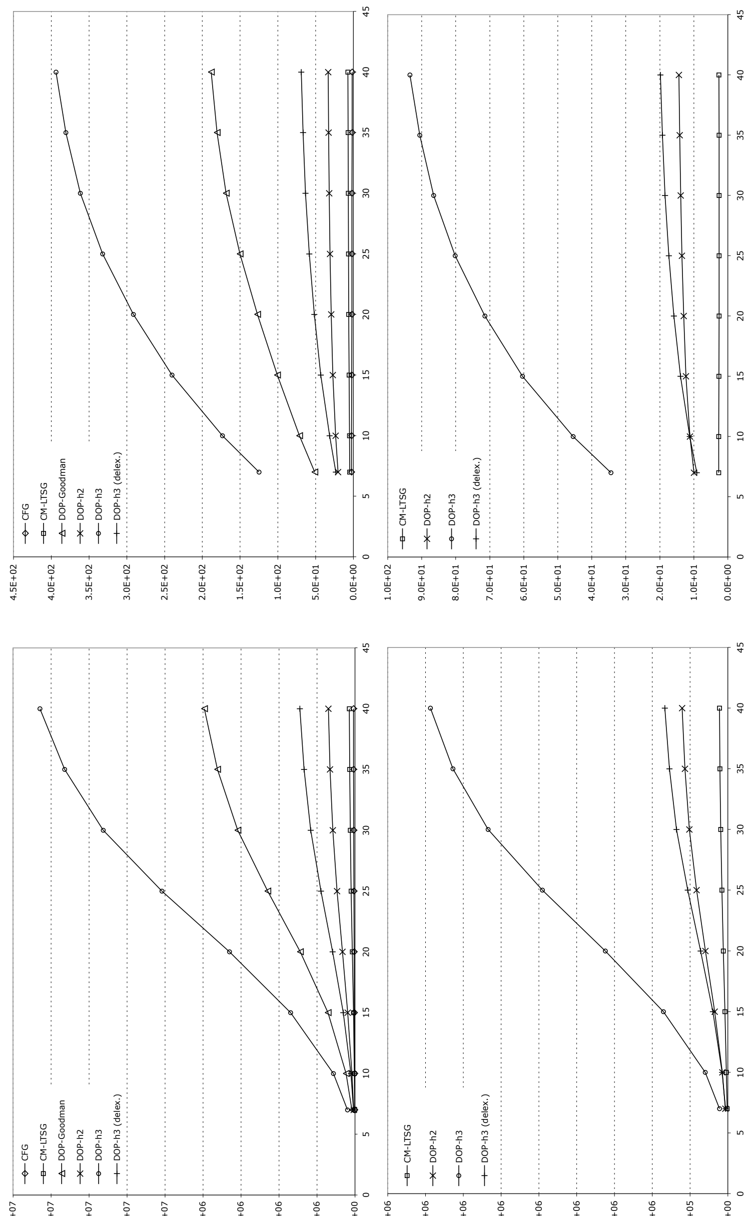
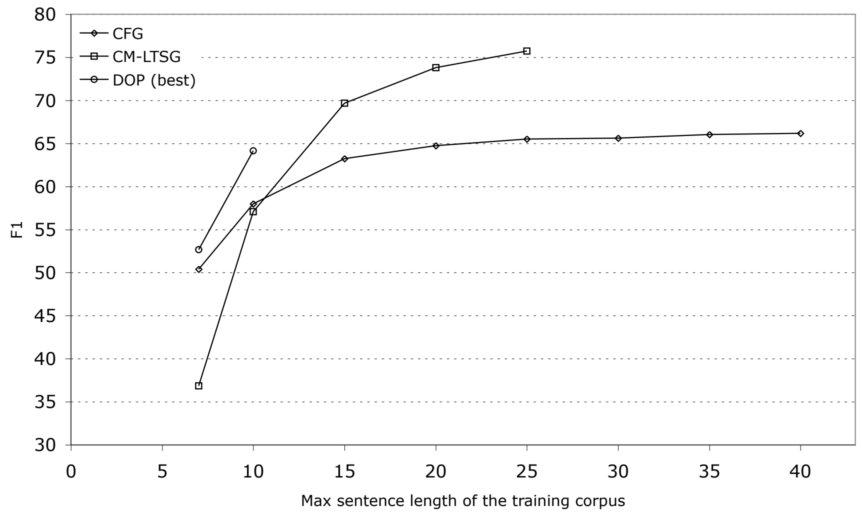


Figure 2.9: Growth of the grammar size in different models for increasing length of the sentences in the training corpus. The two metrics used are the number of CFG rules (above) and the number of elementary trees (below). The two graphs in the right side are a normalization of their left respective ones which take into consideration the distribution of the WSJ for different sentence lengths, i.e. as if the corpus were containing equally abundant sentences for all the length categories. In the bottom graphs DOP-Goodman and CFG are not reported because their elementary trees are equivalent to CFG rules.



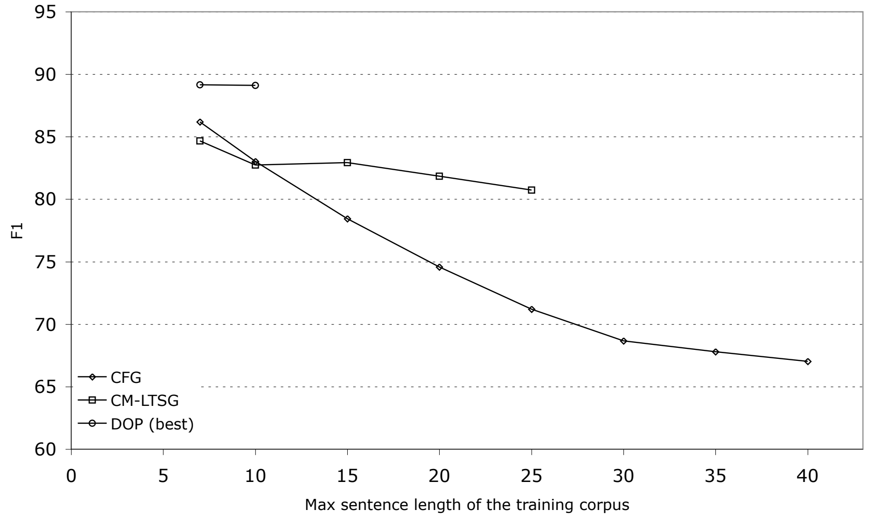


Figure 2.10: F1 scores of the three models (CFG, CM-LTSG, DOP), on WSJ sec22 (for sentences with length less than 60 words above, and for sentences length less than the maximum length of the sentences in the training length below.

2.7.3 Results Analysis

There are two main analyses that we can draw from examining the results. The first concerns the size of the different grammars while the second is about the actual performance of the models.
The size of the grammar has direct impact on the amount of memory and time which is required by the parser to derive the correct analysis of an input sentence. While this could have relevance in real-time applications such as speech-recognition systems, it is not a point of main concern when accounting for the potential accuracy of the grammar (provided there are enough computational resources).

When observing the graphs in Figure 2.9 we notice how the DOP-like approaches are the ones with the biggest grammars: even in its most reduced version (DOP_{h2}), there are nearly 2 million CFG rules which are generated from sentences up to length 40 in the training corpus. This makes evaluating DOP infeasible on easily accesible machines.
It is nevertheless important to mention that even in this most demanding approaches, the growths of the grammars are not exponential with the length of the sentences, since they all show signs of leveling off.
Apart from DOP we observe how the LTSG generated by the head dependency rules of Collins and Magerman is very compact in size. In fact the growth of its grammar is very close to the one relative to the CFG.
The performances of the different models are synthesized in the two graphs of Figure 2.10. In particular the first and second graph in the figure represent a lower and an upper bound for the results respectively. Regarding the DOP approach, although the initial results are very promising, we are unable to evaluate it for sentences longer than 10 words, for the complexity problems mentioned before. Within the other approaches although the LTSG has lower results than the CFG for very short sentences[11], it outperforms the CFG when the grammars are trained with sentences with more than 15 words. In particular, the lower bound on performance drastically decreases for the CFG while it is more stationary for the LTSG.
According to these results we can see that the CFG reaches an F1 score of 66.5% on WSJ40 and we can easily predict that LTSG rules could reach a score on the same target set in between 75% and 80%[12]. This makes the subclass of LTSGs generated by head percolation rules, a promising methodology, both in terms of its encouraging performance and its computational tractable grammar.

[11]It is important to report that while all methods have a very small parse failure, LTSG have a relevant number of failed sentences when trained on WSJ7 (551/1699) and WSJ10 (127/1699).

[12]This prediction will be confirmed in the final results of the next chapters, when we wil develop a simple strategy to further compress this grammar (see section 3.7.1).

Chapter 3

Lexicalized Tree Substitution Grammars

3.1 Introduction

In the previous chapter we described and compared various STSG models. In this section we will focus on a subclass of Lexicalized Tree Substitution Grammars, which considers elementary trees with exactly one lexical anchor.
We saw how the lexicalized grammar generated by the head annotated corpus based on the Collins and Magerman scheme, is very compact in size and yields encouraging result. This scheme is, however, a somewhat arbitrary heuristic, and not ultimately motivated by linguistic theory [Chiang and Bikel, 2002]. This leaves open the challenge to define an automatic way to assign a dependency structure to the parse trees in the training corpus. This would allow to obtain a potentially cross-linguistic methodology that will reduce the human effort to generate such annotations.
We will investigate three different approaches to automatically assign head dependencies to the training corpus: maximum likelihood estimation, entropy minimization, and an alternative heuristic based strategy. The baselines for our experiments will be given by the Collins and Magerman scheme together with the random, the leftmost daughter, and the rightmost daughter based assignment of head dependencies.

3.2 One-anchor LTSGs

The class of one-anchored LTSG is similar to the class of CFGs in the sense that every parse tree consists of a relatively small number of elementary trees, and at the same time resembles the DOP approach in the fact that it allows for fragments of any size to constitute the basic units of the grammar.
In section 2.4.1 we saw that, when considering all possible elementary trees in the

DOP approach, the number of lexical elementary trees and lexical derivations grows exponentially with the size of the tree. The restricted class of grammars that we are considering in this chapter, viz. one-anchor lexicalized trees, presents an interesting property: while the number of lexicalized derivations still grows exponentially with the size of the grammar, the growth of number of elementary trees is polynomial in the number of the lexical items in the parse trees in the worst case and linear in the best case. The graph in Figure 3.1 shows the exact growth in these two cases, the upper bound represented by the most skewed tree and the lower bound by the most balanced one. The new upper bound function ($L'\psi_h = L'\psi_{h-1} + n + 1$) grows in the order n^2. Table 3.1 reports the exact calculation of the number of lexical subtrees and derivations for the new upper and lower bounds.

Skewed Tree	Balanced Tree
$h = n$	$h = \log_2 n + 1$
$L'\psi_h = L'\psi_{h-1} + n + 1$	$L'\psi_h = n \cdot h = n \log_2 n + n$
$N_h = 2 \cdot (h-1)$	$N_h = 2^h - 2$
$P_h = n = h$	$P_h = n = 2^{h-1}$
$L'\delta_h = 2^{N_h+1-P_h}$	$L'\delta_h = 2^{N_h+1-P_h}$

Table 3.1: Calculation of one-anchor lexical subtrees and derivations for balanced and skewed binary trees, where h: depth, n: leaf nodes, $L'\psi$: number of one-anchored lexical subtrees, N: number of internal nodes, P: number of pre-lexical nodes, $L'\delta$: number of lexical derivations using one-anchor lexical elementary trees.

3.3 Building the bag of elementary trees

The first step which will constitute the starting point for some of our experiments (MLE, heuristic) consists of generating all possible one-anchor lexicalized elementary trees from the training corpus. This procedure is similar to the generation of all elementary trees in the DOP approach, and suffers from similar problems of disproportionate distribution of the probability mass. The advantage that we have in the new restricted class of LTSG is that the number of elementary trees is much more modest, and we can successfully deal with all of them.

When considering each parse tree in the training corpus, the extraction procedure takes into consideration one lexical item at a time, and extracts from there, in a bottom up fashion, all the elementary trees anchored in it. If we consider the parse tree of Figure 2.2, and want to extract all the elementary trees anchored in *"Elianti"* we will obtain the following four elementary trees of Figure 3.2.

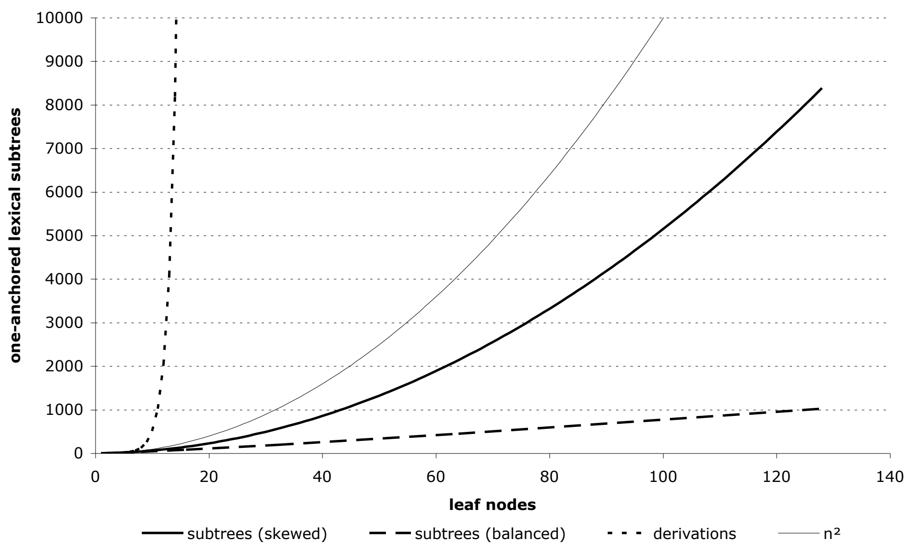


Figure 3.1: Growth of one-anchor lexical subtrees and derivations for balanced and skewed binary trees of increasing number of leaf nodes.

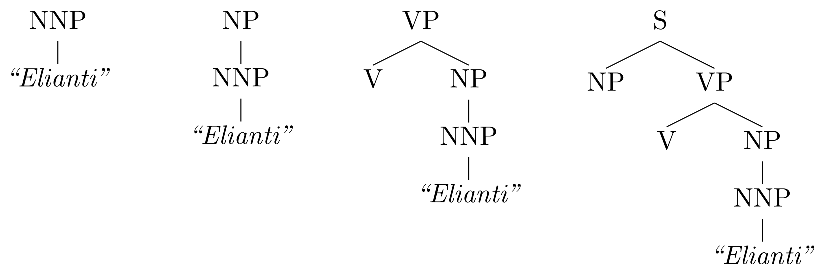


Figure 3.2: Elementary trees anchored in the lexicon *"Elianti"* extracted from the parse tree of Figure 2.2. The first elementary tree is not consistent with the head annotation constraints leading to one-anchor lexicalized grammars; it would in fact require a delexicalized production $NP \rightarrow NNP$ to complete the parse tree.

It's important to understand that the elementary trees we are considering, consist of a simple path, the spine, connecting the root with the lexical anchor, together with all the daughters of the nodes on the spine. This is precisely the way we would extract the same elementary tree if all the nodes of the spine were marked with head dependencies.
As shown in Figure 3.2, when we have unary productions we have chosen to extract all the intermediate elementary trees, even if some of them are not fully consistent with the head annotation constraints leading to one-anchor lexicalized grammars. For instance there is no one-anchor lexicalized derivation of the parse tree of Figure 2.2 which could generate the first elementary tree (NNP *"Elianti"*) reported in Figure 3.2. In fact the pre-lexical NNP is the unique daughter of the NP node, and should therefore always be annotated as head. We notice that this "inconsistent" extraction of elementary trees allows to extract a better grammar when using the heuristic based algorithm. We will use the consistent extraction of elementary trees in the MLE-based algorithm.
If we decide to assign to each extracted elementary tree an equal weight, we will end up with a probability mass distribution problem analogous to the one encountered for the DOP grammars. In fact the number of derivations still grows exponentially with the tree depth, and the majority of the probability mass is therefore reserved to the parse trees of biggest depth. We could apply a Bonnema-like estimator to overcome this problem, but we decided not to go for this direction.

3.4 Maximum likelihood estimation

Once we define the set of all possible elementary trees T in our grammar we need to decide which probability to assign to them. We will denote with $LG : T \rightarrow [0,1]$ the set of all possible probability distributions over T. We have therefore that for every $P \in LG$, $P(\tau) \in [0,1]$. The only constraint we encountered so far is the one defined for the entire class of TSG: the probability of all the elementary tree sharing the same root has to sum up to one. If we indicate with $\mathscr{T}^*$ the set of all possible parse trees that can be built with elements of T, we will have that every $P \in LG$ defines a probability p for every parse tree $t \in \mathscr{T}^*$:

$$p(t) = \sum_{d \in \delta(t)} p(d) = \sum_{d \in \delta(t)} \prod_{\tau \in d} P(\tau)$$

Analyzing the problem from a MLE perspective we will treat the parse trees in our treebank $\mathscr{T} = \langle t_1, t_2, \ldots, t_n \rangle$ as the observed data. To account for the possibility of a parse tree t to occur more than once, we need to define a frequency function $f : \mathscr{T}^* \rightarrow \mathbb{Q}_{\geq 0}$. We are allowing here the frequency of a parse tree to be a weighted (rational) occurrence frequency.
We will denote with $\mathcal{M}_0$ the set of all possible probability distributions (unrestricted model) over $\mathscr{T}^*$, and with $\mathcal{M}_{LG}$ the ones generated by LG:

$$\mathcal{M}_{LG} = \Big\{ p \in \mathcal{M}_0 \mid \exists\, P \in LG\, s.t. \forall t \in \mathscr{T}^*,\; p(t) = \sum_{d \in \delta(t)} \prod_{\tau \in d} P(\tau) \Big\}$$

Finally we define with $L_{\mathscr{T}}(p, f)$ the likelihood of the probability instance $p \in \mathcal{M}_{LG}$ on the corpus $\mathscr{T}$ as follows:

$$L_{\mathscr{T}}(p, f) = \prod_{t \in \mathscr{T}} p(t)^{f(t)} = \prod_{t \in \mathscr{T}} \Big(\sum_{d \in \delta(t)} \prod_{\tau \in d} P(\tau) \Big)^{f(t)}$$

In simple terms, the likelihood function quantifies how well p fits the observed corpus. We are interested to find out the probability distribution $\hat{p}$ which maximizes the likelihood of the training corpus, so that:

$$L_{\mathscr{T}}(\hat{p}, f) = \max_{p \in \mathcal{M}_{LG}} L_{\mathscr{T}}(p, f)$$

Now, thinking about the set of probability distributions $\mathcal{M}_{LG}$, we can certainly confirm that $\mathcal{M}_{LG} \subseteq \mathcal{M}_0$. If we can demonstrate they are equal, we can take the relative-frequency estimate $\tilde{p}$ as the maximum-likelihood estimate.

$$\tilde{p}(t) = \frac{f(t)}{|\mathscr{T}|}$$

In fact, as shown in [Prescher, 2005], the relative-frequency estimate is the unique maximum-likelihood estimate for the unrestricted probability model $\mathcal{M}_0$. The same paper shows that if $\tilde{p}$ is an instance of the model $\mathcal{M}_{LG}$ then it is the unique maximum-likelihood estimate. Unfortunately, as in PCFG, $\tilde{p} \notin \mathcal{M}_{LG}$ if LG is read off from a finite corpus $\mathscr{T}$, and there is a full-parse tree t_∞ which is not in $\mathscr{T}$ but can be generated from our LG. It is very well the case that our set T of elementary trees, upon which every distribution $P \in LG$ is based, is read off from a finite treebank $\mathscr{T}$ and it allows for many more parse trees not present in the training treebank. We have in fact that $\mathscr{T} \subset \mathscr{T}^*$. It follows that our $\mathcal{M}_{LG}$ is a restricted model and that the relative-frequency estimator is not the true maximum-likelihood estimate.

It has been proved [Prescher, 2002] that in PCFG the relative frequency estimator on the grammar rules, induces a probability over the full parse trees in the training corpus, which is exactly the maximum-likelihood estimate in the restricted model. This is due to the fact that CFGs define a unique derivation for every complete parse trees, and maximizing the likelihood in the restricted model is equivalent to maximize the product of the probability of each single rule occurring in each parse tree in the training corpus. As we have already seen, one-anchor lexicalized tree substitution grammars, unlike CFGs, define more than one derivation for every parse tree. We should therefore find an other way to calculate the maximum-likelihood estimate.

3.4.1 EM algorithm

The Expectation-Maximization algorithm [Dempster et al., 1977] is a way to find a maximum-likelihood estimate in circumstances, like ours, where direct estimation is infeasible. As shown before, the likelihood function on one-anchor lexicalized substitution grammars turns out to be defined as a product of sums, and is therefore hard to maximize analytically.

Following [Prescher, 2005], we illustrate the input and the procedure of EM applied to our specific grammar.

The input of the EM algorithm is:

(i) a *set of incomplete-data types* $\mathscr{T}^*$

(ii) an *incomplete-data corpus*, i.e., a frequency distribution f on the set of *incomplete-data types*

$$f : \mathscr{T}^* \to \mathbb{Q}_{\geq 0} \quad \text{such that} \quad 0 < \sum_{t \in \mathscr{T}^*} f(t) < \infty \quad \text{with } \forall t, f(t) \geq 0$$

(iii) a *set of complete-data types* $\mathscr{D}$: the set of derivations $d_1, d_2, \ldots, d_k$ for all possible $t \in \mathscr{T}^*$

(iv) a *symbolic analyzer*, i.e., a function $\delta : \mathscr{T}^* \to 2^{\mathscr{D}}$ which assigns a *set of analyzes* $\delta(t) \subseteq \mathscr{D}$ to each *incomplete-data type* $t \in \mathscr{T}^*$, such that all sets of analyzes form a partition of the set $\mathscr{D}$ of *complete-data types*:

$$\mathscr{D} = \bigcup_{t \in \mathscr{T}^*} \delta(t)$$

(v) a *complete-data model* $\mathcal{M}_{LG} \subseteq \mathcal{M}_0$, i.e., each instance $p \in \mathcal{M}_{LG}$ is a probability distribution on the set of *complete-data types*. The distribution p is implicitly defined by a probability $P : T \to [0,1]$ on the set of elementary trees T.

$$p : \mathscr{D} \to [0,1] \quad \text{and} \quad \sum_{d \in \mathscr{D}} p(d) = 1 \quad \text{and} \quad p(d) = \prod_{\tau \in d} P(\tau)$$

(vi) an *incomplete-data model* $\mathcal{M}_{LG} \subseteq \mathcal{M}_0$ implicitly defined, induced by the *symbolic analyzer* and the *complete-data model*[1]

$$p : \mathscr{T}^* \to [0,1] \quad \text{and} \quad p(t) = \sum_{d \in \delta(t)} p(d)$$

(vii) a *starting instance* $P_0 \in LG$ defining an instance p_0 of the complete-data model $\mathcal{M}_{LG}$

[1]Each probability $p \in \mathcal{M}_{LG}$ defines a probability distribution over the *set of complete-data types* $\mathscr{D}$ and the *set of incomplete-data types* $\mathscr{T}^*$. In fact the latter is a marginal distribution of the former one.

The procedure of the EM algorithm is:

Algorithm 2 *EM procedure*

```
Require: a starting instance $P_0 \in LG$ defining an instance $p_0$ of the complete-data model $\mathcal{M}_{LG}$
  for all $i = 0, 1, 2, \ldots$ do
    E-step: compute the complete-data corpus $f_{\mathscr{D}} : \mathscr{D} \rightarrow \mathbb{Q}$ from $p_i$ and $f$:
            $f_{\mathscr{D}}(d) = f(t_d) \cdot p_i(d|t_d) \quad \text{where} \quad d \in \delta(t_d)$
    M-step: compute the maximum-likelihood estimate $\hat{p} \in M_{LG}$ on $f_{\mathscr{D}}$:
            $L_{\mathscr{D}}(\hat{p}, f_{\mathscr{D}}) = \max_{p \in \mathcal{M}_{LG}} L_{\mathscr{D}}(p, f_{\mathscr{D}})$
            $p_{i+1} = \hat{p}$
  end for
  return $p_0, p_1, p_2, \ldots, p_\infty$
```

The iteration of the EM algorithm is demonstrated to produce a sequence of probability instances $p_0, p_1, p_2, \ldots, p_\infty$ resulting in a monotonically increasing likelihood on the *incomple-data corpus*.

In the E-step the algorithm aims at distributing the observed frequency $f(t)$ of each parse tree t in the training corpus $\mathscr{T}$ to all the derivations $d \in \delta(t)$. In general, each frequency $f(t)$ of a tree t in the set of *incomplete-data types* $\mathscr{T}^*$ will be distributed among its derivations $\delta(t) = d_1, d_2, \ldots, d_{|\delta(t)|}$, based on the conditional probability $p(d_i|t)$. This distribution generates the *complete-data corpus* $f_{\mathscr{D}}$.
In the M-step we calculate the MLE on the *complete-data corpus* $f_{\mathscr{D}}$. The likelihood on the complete-data corpus is defined as:

$$L_{\mathscr{D}}(p, f_{\mathscr{D}}) = \prod_{d \in \mathscr{D}} p(d)^{f_{\mathscr{D}}(d)}$$

Maximizing the likelihood of the *complete-data corpus* is much more simple than maximizing the likelihood of the *incomplete-data corpus*. The new likelihood is in fact a simple product, and its maximization, is trivially given with relative frequency on the set of elementary trees extracted from $\mathscr{D}$ based on $f_{\mathscr{D}}$.
In the proof which follows we will denote with $f_T(\tau, d)$ the number of times the fragment τ occurs in the derivation d, with $f_T(\tau)$ the frequency of $\tau \in T$ in all the derivations $\mathscr{D}$ weighted by the frequency of each d where it occurs, and with T_A the subset of T containing only fragments with category A.

$$
\begin{aligned}
\max_{p \in \mathcal{M}_{LG}} L_{\mathscr{D}}(p, f_{\mathscr{D}}) &= \max_{p \in \mathcal{M}_{LG}} \prod_{d \in \mathscr{D}} p(d)^{f_{\mathscr{D}}(d)} \\
&= \max_{P \in LG} \prod_{d \in \mathscr{D}} \Big(\prod_{\tau \in d} P(\tau) \Big)^{f_{\mathscr{D}}(d)} \\
&= \max_{P \in LG} \prod_{d \in \mathscr{D}} \Big(\prod_{\tau \in T} P(\tau)^{f_T(\tau, d)} \Big)^{f_{\mathscr{D}}(t)} \\
&= \max_{P \in LG} \prod_{d \in \mathscr{D}} \prod_{\tau \in T} P(\tau)^{f_T(\tau, d) \cdot f_{\mathscr{D}}(d)} \\
&= \max_{P \in LG} \prod_{\tau \in T} \prod_{d \in \mathscr{D}} P(\tau)^{f_T(\tau, d) \cdot f_{\mathscr{D}}(d)} \\
&= \max_{P \in LG} \prod_{\tau \in T} P(\tau)^{\sum_{d \in \mathscr{D}} f_T(\tau, d) \cdot f_{\mathscr{D}}(d)} \\
&= \max_{P \in LG} \prod_{\tau \in T} P(\tau)^{f_T(\tau)} = \max_{P \in LG} L_T(P, f_T) \\
&= \max_{P \in LG} \prod_{A} \prod_{\tau \in T_A} P(\tau)^{f_T(\tau)} \\
&= \max_{P \in LG} \prod_{A} \prod_{\tau \in T_A} P(\tau)^{f_{T_A}(\tau)} \\
&= \max_{P \in LG} L_T(P, f_{T_A})
\end{aligned}
$$

The proof reported above[2], shows that the MLE function on *complete-data corpus* $f_{\mathscr{D}}$ is equivalent to the MLE defined on the fragments T extracted from $\mathscr{D}$ with the corresponding weighted frequency $f_{\mathscr{D}}$. Moreover, the last three lines of the equation prove that this MLE equals the relative-weighted-frequency estimation of each fragment T in the set of fragments sharing the same root.
This proof suggests to combine the E and the M step in a unique process, which extracts on the fly the weighted frequency of the fragments in T as shown in Algorithm 3.

The EM algorithm is demonstrated to converge to a *local maximum-likelihood estimate*. The convergence to the *global maximum* depends on the initialization of the algorithm, i.e. the probability distribution $P_0 \in LG$ assigned to our fragments T. It is always recommended to choose a P_0 such that $P(\tau) > 0$ for all $\tau \in T$. We want to try out two different strategies: in the first one we assign a uniform distribution to all the fragments sharing the same category (EM_U), and in the second one (EM_{DOP}) we use a DOP-like estimation of the frequency of each elementary tree, as describe in section 3.3.
We should also remember that our goal, once we assign a probability to each elementary tree, is to annotate each parse tree in the training corpus with head dependency, to define the ultimate set of elementary trees constituting our final grammar. Although more than one derivation is possible for each training parse tree, we will therefore only choose the one with maximum probability.

[2]This proof is very similar to the one shown for the case of PCFG in [Prescher, 2005].

Algorithm 3 *EM procedure (compact)*

```
Require: a starting instance $P_0 \in LG$ defining an instance $p_0$ of the complete-
    data model $\mathcal{M}_{LG}$
  for all $i = 0, 1, 2, \ldots$ do
    Initialization: initialize to 0 all frequency of all fragments in $T$: for all
    $\tau \in T$, $f_T(\tau) = 0$
    EM-step: extract on the fly the weighted frequency of the fragments in $T$
    for each derivation $d \in \mathscr{D}$:
    for all $t \in \mathscr{T}$ do
      calculate $p_i(t) = \sum_{d \in t} \prod_{\tau \in d} P(\tau)$
      for all $d \in \delta(t)$ do
        update† $f_{\mathscr{D}}(d) \overset{+}{\leftarrow} p_i(d|t_d) = \frac{\prod_{\tau \in d} P_i(\tau)}{p_i(t)}$
      end for
    end for
          $P_{i+1} = \hat{P} : \hat{P}(\tau) = \frac{P_i(\tau)}{\sum_{\tau' : r(\tau') = r(\tau)} P_i(\tau')}$
  end for
  return $P_0, P_1, P_2, \ldots$
```

† In the algorithm, when updating the *complete-data corpus* $f_{\mathscr{D}}$, we omitted the frequency $f(t_d)$ of the observed *incomplete-data type* t_d since we consider $\mathscr{T}$ as a collection of parse trees and we treat each element as if it occurs once.

3.4.2 EM *n-best* implementation

When implementing the EM procedure of Algorithm 3, one has to face the problem of considering all possible derivations of each parse tree in the training corpus. As shown in Figure 3.1 this number grows exponentially with the length of the sentence yielded by the parse tree. We therefore decide to implement an *n-best* approximation of the algorithm considering the *n-best derivations* of each parse tree. In order to adopt this strategy, we need to define a dynamic algorithm which, in a bottom-up fashion, selects the n-best derivations of each node in the parse tree under consideration. Algorithm 4 describes how to build the tree-structure of tables for each parse tree, while Algorithm 5 describes how each table is computed. For defining the two algorithms we will use the following notation:

Notation	
$depth(t)$	the maximum depth of t
$\mathscr{I}_h(t)$	the set of non-terminal nodes N in t of depth h
$\mathscr{L}(N)$	the set of lexicons under N, $\mathscr{L}_i(N)$ being the i-th lexical entry
$\tau_i(N)$	the elementary tree rooted in N anchored in $\mathscr{L}_i(N)$
$P(\tau_i(N)$	the probability assigned to the fragment $\tau_i(N)$ by P
$S(\tau_i(N))$	the set of substitution sites in $\tau_i(N)$, $S_s(\tau_i(N))$ is the s-th site
$I_j(N)$	indices, defining the j-th best derivation rooted in N: let $\tau_i(N)$ be the tree starting the derivation, $I_j(N)$ is the set of $\|S(\tau_i(N))\|$ indexes, where $I_j^s(N)$ is the index specifying which derivation to use in the $S_s(\tau_i(N))$-th substitution site of $\tau_i(N)$ among its n-best derivations $\delta(S_s(\tau_i(N)))$
$\delta(N)$	the set of n-best derivations rooted in N, where the j-th best derivation $\delta_j(N)$ is represented as $\{\tau_i(N), I_j(N)\}$
$p(\delta_j(N))$	the probability of the j-th derivation $\delta_j(N)$ for which $\exists\, i$ s.t. $\delta_j(N) = \{\tau_i(N), I_j(N)\}$ and $p(\delta_j(N)) = P(\tau_i(N)) \cdot \prod_{s=1}^{\vert S(\tau_i(N))\vert} P(\delta_{I_j^s(N)}(S_s(\tau_i(N))))$

3.4.3 Overfitting

The EM procedure described before, is the same one which could generate the MLE on the full set of elementary trees, constituting the grammar of the general DOP approach. It has been demonstrated [Zollmann and Sima'an, 2005] that in this case MLE converges to the relative frequency $\hat{p}$ of the parse trees in the training corpus. This derives from the fact that the bag of elementary trees contains a full parser tree instance, for every parse tree t in the training corpus. It follows that there exist a probability distribution of the set of elementary trees which assign to each parse tree t in the input corpus a probability which equals its relative frequency $\hat{p}(t)$: this probability distribution assigns to every full parse tree t in T the probability $\hat{p}(t)$, and to all the other fragments arbitrary weights. Since the relative frequency estimator is in our model, it coincides with the unique MLE. This is not a desirable result, since the probabilistic grammar associated with the MLE is not able to generalize over unseen sentences. In other terms the grammar shows a maximum degree of *overfitting*.
Unlike DOP, LTSGs don't allow full parse trees to be in the elementary tree collection. This is the main reason why the proof synthesized before is not applicable for our LTSGs. Nevertheless in our experiments, we do encounter a certain degree of overfitting: as shown in Figure 4.1c, the elementary trees which receive higher probability after an EM training are usually very deep, and for this reason they loose the ability to be general enough to parse new sentences.

Algorithm 4 *n-best_derivations*(t, n)

```
Require: $t$ is a parse tree
  for all $h = depth(t) - 1$ down to 0 do
    for all $N \in \mathscr{I}_h(t)$ do
      if $\mathscr{L}(N)| = 1$ then
        $\delta_1(N) = \{\tau_1(N), null\}$
        $p(\delta_1(N) = P(\tau_1(N))$
      else
        for all $i = 1, \ldots, |\mathscr{L}(N)|$ do
          extract $\tau_i(N), S(\tau_i(N))$
          $update\_n\text{-}best\_delta(\delta(N), p(\delta(N)), S(\tau_i(N)), \tau_i(N), P(\tau_i(N)))$
        end for
      end if
    end for
  end for
  $N_{TOP} = root(t)$
  return $\delta(N_{TOP})$
```

Algorithm 5 *update_n-best_table*$(\delta(N), p(\delta(N)), S(\tau_i(N)), \tau_i(N), P(\tau_i(N)))$

```
Require: $\delta(N)$ is ordered such that $p(\delta_1(N)) > p(\delta_2(N)) > \ldots > p(\delta_n(N))$
  initialize $tmp\_I$: $tmp\_I^s = 1$ for all $s = 1$ to $|S(\tau_i(N))|$
  {$tmp\_I$ is the set of indexes defining one of the possible derivations starting with $\tau_i(N)$;
  $I^s$ is the index among the n-best derivations of the $s$-th substitution site of $\tau_i(N)$ used
  in the current derivation. $tmp\_I$ is initialize by choosing the best sub-derivation for each
  substitution site of $\tau_i(N)$}
  while TRUE do
    $N_s = S_s(\tau_i(N))$ {$N_s$ is the $s$-th substitution site of $\tau_i(N)$}
    $temp\_prob = P(\tau_i(N)) \cdot \prod_{s=1}^{|S(\tau_i(N))|} P(\delta_{tmp\_I^s}(N_s)$
    if $temp\_prob > \delta_n(N)$ then
      determine $x$ such that $\delta_{x-1}(N) > temp\_prob > \delta_{x+1}(N)$
      $I_x(N) = tmp\_I$
      $\delta_x(N) = \{\tau_i(N), I_x(N)\}$
      $P(\delta_x(N)) = temp\_prob$
    else
      return {the are no more derivations starting with $\tau_i(N)$ which are within the n-best
      derivations rooted in $N$}
    end if
    {find index $m$ such that}
    $m = \max_{s=1,\ldots,|S(\tau_i(N))|} P(\delta_{tmp\_I^s+1}(N_s))$
    if $m = null$ then
      return {there are not other derivations}
    end if
    $tmp\_I^m \overset{+}{=} 1$
  end while
```

3.5 Entropy minimization

EM aims at maximizing the likelihood of the model. An other criteria which is the target of other machine learning techniques is the simplicity of the data. In this session we will describe an entropy based algorithm, which aims at learning the most "simple" grammar fitting the data.
Specifically we aim at reducing the uncertainty of the structures which can be associated to each lexical elements.We achieve this by minimizing an objective function which relates to an entropy measure of our grammar. This function is based on the general definition of entropy in information theory. If we have a discrete stochastic variable X taking n possible values $x_1, x_2, \ldots, x_n$ with probabilities $p(x_1), p(x_2), \ldots, p(x_n)$ respectively, we will define the entropy of X as:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

We chose to measure the sparseness of our grammar, by grouping all the elementary trees which share the same unique lexical anchor. For every lexical item $l \in \mathscr{L}$ we define the discrete stochastic variable T_l as the set of all the elementary trees having l as lexical anchor $\tau_{l_1}, \tau_{l_2}, \ldots, \tau_{l_n}$. We will then have an estimation of the entropy of our grammar by summing the entropy of each single discrete stochastic variable T_l for each occurrence of l. If $f(\tau)$ and $f(l)$ are the frequency of the fragment τ and the lexical item l in the head annotated corpus, the entropy H of our grammar is defined as:

$$H(T) = -\sum_{l=1}^{|\mathscr{L}|} \sum_{i=1}^{n} p(\tau_{l_i}) \log_2 p(\tau_{l_i})$$
$$\text{where } p(\tau_{l_j}) = \frac{f(\tau_{l_j})}{\sum_{i=1}^{n} f(\tau_{l_i}))} = \frac{f(\tau_{l_j})}{f(lex(\tau_{l_j}))}$$

In order to minimize the entropy of our grammar, we will apply a *hill-climbing* strategy. The algorithm starts from an already annotated tree-bank (for instance using a random annotator) and iteratively tries to find a change in the annotation of each parse tree which reduces the entropy of the entire grammar, until no further modification which could reduce the entropy is possible. Since the entropy measure is defined as the sum of the function $p(\tau) \log_2 p(\tau)$ of each fragment τ, we don't need to re-calculate the entropy of the entire grammar, when modifying the annotation of a single parse tree. In fact:

$$\begin{aligned} H(T) &= -\sum_{l=1}^{|\mathscr{L}|} \sum_{i=1}^{n} p(\tau_{l_i}) \log_2 p(\tau_{l_i}) \\ &= -\sum_{j=1}^{|T|} p(\tau_j) \log_2 p(\tau_j) \end{aligned}$$

$$\text{where } p(\tau_j) = \frac{f(\tau_j)}{f(lex(\tau_j))}$$

Although there is no guarantee to find the annotation with the absolute minimum entropy, the algorithm is very efficient and succeeds to drastically minimize the entropy from a random annotated corpus. There are few parameters which can be set to change the behavior of the algorithm: *CHANGES*: the maximum number of repetitive changes in the same parse tree; *ORDER*: whether, among all possible changes which would reduce the entropy of an input parse tree, we want to choose a random one, the one leading to the maximum reduction of the entropy, or the one leading to minimal reduction; *MIN_THRESHOLD*: as the minimum reduction of entropy required for a particular change in the annotation of a parse tree. Although not all possible combinations of the parameters where evaluated, for the ones we tried, slightly different results were detected. The general setting which are used in the final evaluations are: $CHANGES = 1$, $ORDER = random$, $MIN\_THRESHOLD = 0.01$.
Algorithm 6 presents a pseudo-code of this version of the hill-climbing algorithm. For each input parse tree under consideration, the algorithm tries to randomly modify a head annotation. For a random internal node N, we try to annotate the current head daughter H as the new dependent, and a different daughter D of N as the new head. When we consider the changes that this modification brings on the set of the elementary trees T, we understand that there are only 4 elementary trees affected:

- $\tau_h(N)$ (*old head tree*): the lexical tree, including N and H, with H being the head of N.
- $\tau_d(N)$ (*old dependent tree*): the lexical tree, rooted in D, with D being the dependent of N.
- $\tau'_d(N)$ (*new dependent tree*): the lexical tree, rooted in H, with H being the dependent of N.
- $\tau'_h(N)$ (*new head tree*): the lexical tree, including both N and D, with D being the head of N.

We used the terms *old* and *new* to refer to the role of the trees before and after the change respectively. After making the change in the head annotation, we just need to decrease the frequencies of the *old trees* by one unit, and increase the ones of the *new trees* by one unit. The change in the entropy of our grammar can therefore be computed by calculating the change in the partial entropy of these four elementary trees before and after the change.

3.5.1 Preliminary results

We would like to test how the notion of entropy we defined is able to provide a valid measure of the quality of our grammar. We decide to run two preliminary experiments, relating the entropy of our grammar with the F_1 score on parsing

Algorithm 6 *hill-climbing*($\mathscr{T}, f_T, f_{\mathscr{L}}$)

```
Require:
    $\mathscr{T}$ is the input treebank already head annotated
    $f(\tau)$ is the frequency of $\tau$ in the bag of elementary trees $T$
    $f(l)$ is the frequency of the lexicon $l$ in the bag of elementary trees $T$
  $MIN\Delta = 0.01$
  $CONTINUE = TRUE$
  while CONTINUE do
    $CONTINUE = FALSE$
    for all $t \in \mathscr{T}$ do
      $MODIFIED = FALSE$
      for all internal non-prelexical and non-unary nodes $N$ of $t$ do
        if MODIFIED then
          break
        end if
        $H$ = the head annotated daughter of $N$
        extract $\tau_h(N)$ {old head tree}
        extract $\tau'_d(N)$ {new dependent tree}
        for all $D \neq H$ daughters of $N$ (randomly selected) do
          annotate $D$ as the new head
          annotate $H$ as the new dependent
          extract $\tau'_h(N)$ {new head tree}
          extract $\tau_d(N)$ {old dependent tree}
          $l_1 = lex(\tau_h(N)) = lex(\tau'_h(N))$
          $l_2 = lex(\tau_d(N)) = lex(\tau'_d(N))$
          $H0 = \frac{f(\tau_h)}{l_1} \log_2 \frac{f(\tau_h)}{l_1} + \frac{f(\tau'_h)}{l_1} \log_2 \frac{f(\tau'_h)}{l_1} + \frac{f(\tau_d)}{l_1} \log_2 \frac{f(\tau_d)}{l_2} + \frac{f(\tau'_d)}{l_1} \log_2 \frac{f(\tau'_d)}{l_2}$
          $f(\tau_h) = f(\tau_h) - 1,\ f(\tau_d) = f(\tau_d) - 1$
          $f(\tau'_h) = f(\tau'_h) + 1,\ f(\tau'_d) = f(\tau'_d) + 1$
          $H1 = \frac{f(\tau_h)}{l_1} \log_2 \frac{f(\tau_h)}{l_1} + \frac{f(\tau'_h)}{l_1} \log_2 \frac{f(\tau'_h)}{l_1} + \frac{f(\tau_d)}{l_1} \log_2 \frac{f(\tau_d)}{l_2} + \frac{f(\tau'_d)}{l_1} \log_2 \frac{f(\tau'_d)}{l_2}$
          $\Delta H = H0 - H1$
          if $\Delta H > MIN\Delta$ then
            $MODIFIED = TRUE$
            $CONTINUE = TRUE$
          else
            annotate back $H$ as the head of $N$
            annotate back $D$ as the dependent of $N$
            $f(\tau_h) = f(\tau_h) + 1,\ f(\tau_d) = f(\tau_d) + 1$
            $f(\tau'_h) = f(\tau'_h) - 1,\ f(\tau'_d) = f(\tau'_d) - 1$
          end if
        end for
      end for
    end for
  end while
```

test sentences. In the first experiment we measure the entropy of the grammar extracted from 20 random annotations of the input corpus, while in the second we run the hill climbing algorithm, measuring the results at each cycle of the algorithm. Figure 3.3 plots the results from these two experiments. The results of the first experiment suggest that there need not to be a strong correlation between the entropy measure of the grammar and the quality of the grammar, while the second one revels that reducing the sparseness of the the grammar is a way to improve its performance.

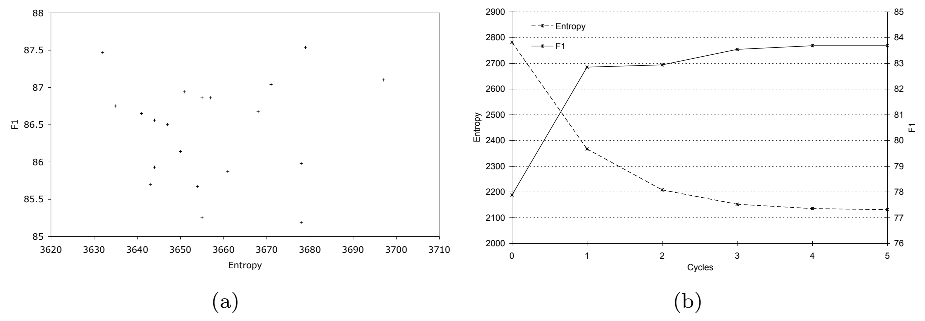


Figure 3.3: (a) Plot of F_1 score over entropy of 20 runs. In each run we start from a randomly annotated input corpus, and calculate the entropy and performance of the extracted elementary trees. (b) Plot of F_1 score over entropy of the grammar annotated with right-annotations, after running the hill climbing algorithm for 0, 1, 2, 3, 4, 5 cycles. In both experiment the training corpus is sec 0-21 of WSJ10 while the test corpus is sec 22 of WSJ10.

3.6 Heuristic based strategy

The main intuition behind the heuristic approach of this section, is to give priority to elementary trees which occur often in our collection of fragments. This is the same as to say that we would like to use elementary trees which are general enough to occur in many possible constructions.
With this idea in mind, after having built the bag of all possible one-anchor lexicalized elementary trees from the training corpus, we need to decide how to annotate each input parse tree with head dependencies.
Two greedy algorithms are here presented. They are similar in the sense that they share the same strategy: for each node we choose to assign the head to the daughter which maximizes the frequency of the derived fragment in the bag of elementary trees. We can compare the occurrences of the different trees, since we know that they share the same root note, and therefore in this case maximizing the relative frequency is equivalent to maximizing their occurrence frequency.
The two strategies differ in the direction they take. The first version (*top*) as-

signs head dependencies starting from the root of the sentence in a top-down fashion, while the second version (*bottom*) uses a bottom up strategy. Algorithm 7 and Algorithm 8 report a pseudo-code for the first and the second version respectively. In many cases the two algorithm generate the same output. Figure 3.4 shows an example of parse tree where the first and the second algorithm generate different head assignments. While the Top-Down strategy relies only on consistent elementary trees, in many cases the Bottom-Up strategy takes advantage of the (inconsistent) intermediate elementary trees. This is the reason why this second version yields better results. The two versions of the algorithm would in fact produce the same exact annotation if we only consider the consistent way of building the bag of elementary trees.

Algorithm 7 *Greedy_Top-Down(N)*

```
Require: $N$ is an internal node of a parse tree
  $L = null$;
  $MAX = -1$;
  for all leaves $l$ under $N$ do
    $\tau_l^N$ = elementary tree rooted in $N$ and anchored in $l$;
    $F$ = frequency of $\tau_l^N$;
    if $F > MAX$ then
      $L = l$;
      $MAX = F$;
    end if
  end for
  Annotate $\tau_L^N$ with head dependency;
  for all non terminal leaves $N_i$ in $\tau_L^N$ do
    $GreedyTopDown(\tau_L^N)$;
  end for
```

Algorithm 8 *Greedy_Bottom-Up(T)*

```
Require: $T$ is a parse tree
  for $i = maxDepth(T) - 1$ down to 0 do
    for all non terminal $N$ of depth $i$ do
      if $N$ has one daughter then
        if $N$ is not pre-lexical then
          mark $N$ as head;
        end if
        CONTINUE;
      end if
      $H = null$
      $MAX = -1$
      for all daughters $D$ of $N$ do
        $\tau_D^N$ = elementary tree rooted in $N$ and passing through $D$ and the
        head dependencies below $D$;
        $F$ = frequency of $\tau_D^N$;
        if $F > MAX$ then
          $H = D$;
          $MAX = F$;
        end if
      end for
      Annotate $\tau_H^N$ with head dependency;
    end for
  end for
```

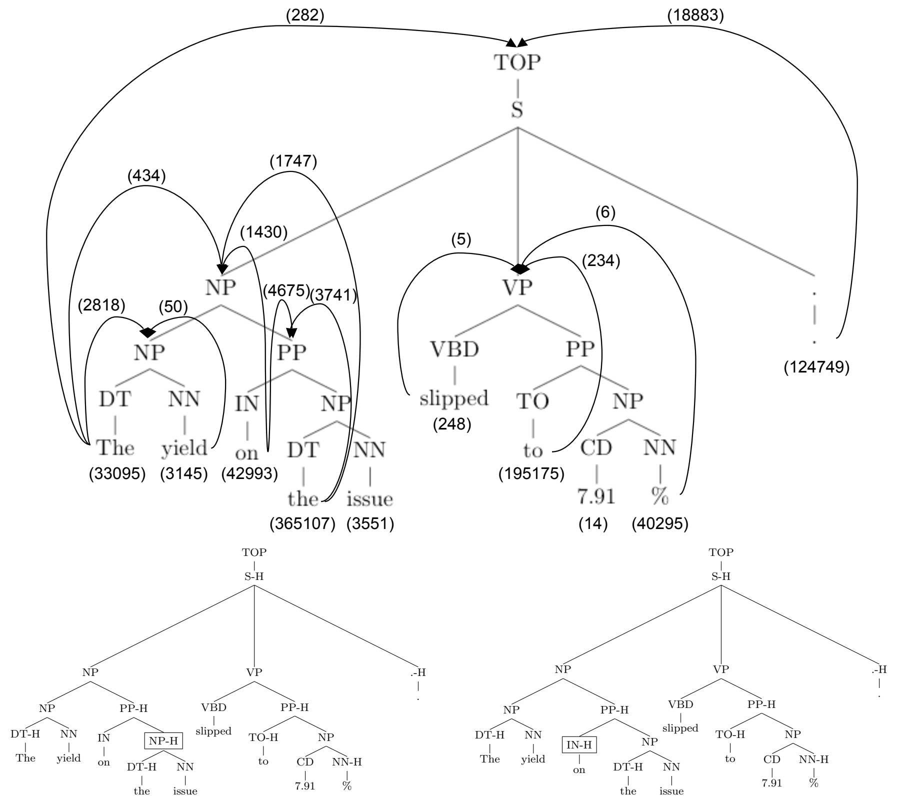


Figure 3.4: Example of a parse tree in the training corpus. The arrows, connecting the root and the anchor of an elementary tree, sketch the most significant elementary trees among the ones which are extracted more than once from the input corpus. The numbers in parenthesis refer to the number of occurrences of the corresponding trees. The number below each lexicon refers to the total number of extracted elementary trees anchored in that lexicon. For instance we have a total number of 282 elementary trees rooted in TOP and anchored in *The*, while the anchor occurs in 33095 elementary trees. The two parse trees in the bottom represent the two head annotations resulting from the *Greedy_Top-Down* and *Greedy_Bottom-Up* algorithm respectively. The difference is situated on the lexical anchors *on* and *the*, the latter prevailing when considering the trees rooted on NP (in the Top-Down algorithm), and the first one outperforming among the trees rooted on NP (in the Bottom-Up algorithm).

3.7 Running the experiments

3.7.1 Removing recursive structures

In section 2.4.2 we defined a way to extract one-anchored lexicalized elementary trees from a head annotated corpus of parsed sentences. If we have a closer look at the biggest elementary trees extracted from the Collins-Magerman head annotation, we notice that many present recursive sentence constructions of the kind illustrated in Figure 3.5.

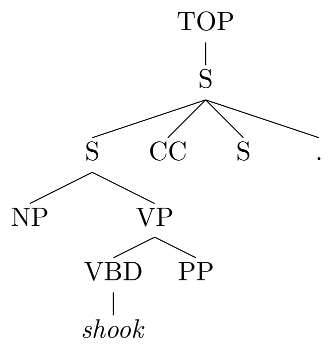
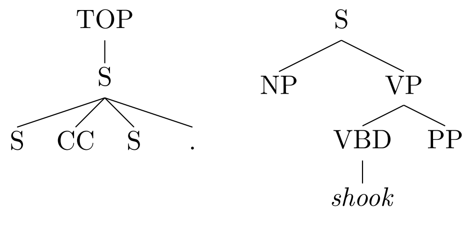


Figure 3.5: One example of recursive structure in an elementary tree taken from the sentence *"This building shook like hell and it kept getting stronger."* The head annotation originating this elementary tree is not specified, but it can be easily retrieved by identifying the spine connecting the TOP node with the lexical anchor.

The trees presenting such recursive structure are constituted by an S under the root of the tree, rewriting to at least an other S. From now on, in our experiment we will break the recursive structure of this elementary trees, in order to allow a more compact grammar, and a generalization of those constructions. As an example the tree reported in Figure 3.5 is split in the following two trees:



After breaking down all the sentences presenting such type of recursion, our parser is able to deal with longer sentences: while before we were only able to train our parser with sentences up to length 25, we can now successfully go up to length 40. Moreover the performance slightly increases for sentences of length above 15.

Furthermore we discovered that in the Collins and Magerman head annotation, there are few instances of parse trees, like the one reported in Figure 4.1a, where unary productions are not head annotated (in the example in the figure the S is the unary production of $SBAR$). This violates our constraint defined for one-anchor lexicalized substitution grammar, in which exactly one daughter of each internal node should be head annotated. As reported in the next section, the corrected version doesn't lead to better results.

3.7.2 Results

We will now compare the results led by the various ways of extracting LTSGs described throughout this chapter. Table 3.2 reports the performances on WSJ section 22 of the three approaches together with 4 baseline strategies. We consider both the case in which the trees in the grammars are delexicalized and lexicalized and whether we are using or not the extra *typology tags* (after the dash sign) such as SBJ, LOC, TMP. In our baseline experiments we considered the Collins and Magerman annotation (*CM-LTSG*) described in 2.4.2, *Random-LTSG*, which is obtained with a random assignment of head annotations, *Left-LTSG* where the leftmost daughter is always marked as head, and *Right-LTSG* similarly for the the rightmost daughter.
While the Collins-Magerman scheme results in a quite compressed grammar due to the language regularities caught by the dependencies rules, these three "naive" approaches cannot find the same degree of regularity and as a consequence their grammar size is too large to be handled for the parser.
In these and other experiments (*EM*, *Entropy*) where this problem arises, we decide to remove from our grammar all the elementary trees occurring once, and then apply a smoothing technique based on the simple CFG formalism. Specifically we will add to our grammar all the CFG rules that can be extracted from the training corpus and give them a small weight proportional to their frequency[3]. This in general will ensure coverage, i.e. that all the sentences in the test set can be successfully parsed. The CFG rules are in fact supposed to "fill the holes" when the lexicalized grammar is not able to produce a complete parse tree of a test sentence. In practice this smoothing technique tends to slightly improve the results. In order to better compare the results we will also show the results when using the same smoothing technique for *CM-LTSG* and the other compact grammars.

[3]In our implementation, each CFG rule frequency is divided by a factor 100.

Delexicalized

	S	***LR***	***LP***	***F_1***	***H***	***|T|***	***|T'|***
Greedy (Bot.)	No	76.39	78.08	77.22	12K	77K	
Greedy (Top)	No	76.22	78.00	77.10	12K	76K	
Left	No	76.29	76.69	76.49	19K	85K	
Random	Yes	77.16	78.59	77.87	25K	120K	71K
Entropy (Rand.)	Yes	77.25	78.04	77.64	11K	125K	69K
Greedy (Bot.)	Yes	76.60	78.25	77.41	12K	77K	68K
Greedy (Top)	Yes	76.48	78.29	77.37	12K	76K	68K
Left	Yes	76.41	76.51	76.46	19K	85K	67K
EM_U (1-best)†	Yes	69.20	67.21	68.19	130K	64K	51K
EM_{DOP} (1-best)†	Yes	67.68	66.59	67.13	25K	129K	61K
Right	Yes	65.80	65.49	65.64	20K	115K	63K

Delexicalized (with typology tags)

	S	***LR***	***LP***	***F_1***	***H***	***|T|***	***|T'|***
Entropy (CM)	No	77.23	78.17	77.64	16K	90K	
CM	No	76.43	77.13	76.77	23K	84K	
CM (Corr.)	No	76.44	77.05	76.74	85K	72K	
Entropy (CM)	Yes	77.34	78.14	77.74	16K	90K	67K
CM	Yes	76.45	76.87	76.66	23K	84K	67K

Lexicalized

	S	***LR***	***LP***	***F_1***	***H***	***|T|***	***|T'|***
Greedy (Bot.)	Yes	79.40	79.93	79.67	12K	113K	72K
Greedy (Top)	Yes	79.25	79.91	79.58	12K	110K	71K
Random	Yes	79.66	79.38	79.52	25K	218K	69K
Entropy (Rand.)	Yes	79.38	79.34	79.36	11K	157K	59K
Left	Yes	75.14	74.80	74.97	19K	151K	64K
EM_{DOP} (1-best)†	Yes	67.62	70.11	68.84	25K	158K	42K

Lexicalized (with typology tags)

	S	***LR***	***LP***	***F_1***	***H***	***|T|***	***|T'|***
Entropy (Rand.)	Yes	80.72	80.08	80.40	12K	182K	61K
Greedy (Bot.)	Yes	80.42	80.05	80.23	14K	141K	76K
Random	Yes	80.21	79.47	79.84	26K	243K	68K
Entropy (CM)	Yes	79.98	79.46	79.72	16K	146K	59K
Entropy (Left)	Yes	80.01	79.40	79.70	14K	160K	62K
CM	Yes	76.46	76.80	76.63	23K	175K	65K
Left	Yes	76.65	76.48	76.57	20K	176K	66K

Table 3.2: Results on various LTSG on sec 22 of WSJ40. S: whether the smoothing technique with CFG rules is applied or not; *LR*: Labeled Bracketing Recall; *LP*: Labeled Bracketing Precision; F_1: harmonic mean between *LR* and *LP*; *H*: entropy measured on the lexicalized elementary trees *T*; $|T|$: size of the bag of elementary trees after being pre-lexicalized; $|T'|$: size of the bag of elementary trees after removing the fragments occurring once (where smoothing applies); $|R|$: size of the pre-lexical CFG grammar given to the parser. The last three values are given in thousands (K).
† The results relative to *EM-LTSG* are given when implementing the 1-best approximation (Viterbi-Best). We were not able to run better approximation on WSJ40 for memory problems. We believe that the results would not improve with better approximations, since different experiments on WSJ20 using 1-best, 10-best and 100-best approximation lead to very close results.

Chapter 4

Conclusions

4.1 Introduction

We ended the previous chapter with some quantitative results, by providing different one-anchored LTSG performances on parsing new sentences. The general conclusion that we can derive from those results is that within the class of one-anchored LTSG, there are three strategies that have the best performance, namely *Random-LTSG*, *Greedy-LTSG*, *Entropy-LTSG*. Their score is around 78% in the delexicalized version and 80% in the lexicalized version. Surprisingly, the strategy following Collins and Magerman head assignment (*CM-LTSG*) scores worse than these three and not much better than the *Left-LTSG* strategy.

At this point, the central question we would like to answer is whether there are possible ways of achieving better results using the same one-anchored LTSG formalism, or whether the formalism itself is blocking the way to more successful results. This question turns out to be very difficult to answer at this stage: although many different strategies where applied, we were not able to exhaust the entire domain of possible one-anchored LTSGs. Nevertheless, it will appear soon clear that even if better one-anchored LTSGs can be extracted from the input corpus, they would hardly be able to generalize perfectly over new constructions.
For this reason, the goal of this conclusive chapter is to try to understand as much as possible how to evaluate the generalization power of different one-anchored LTSGs through a more qualitative analysis. To do so we will investigate the nature of the basic units of the different grammars, and propose a possible solution to overcome the generalization problems.

In order to have a first intuition on the qualitative difference between the various one-anchored LTSGs, we report in Figure 4.1 the annotations of the used strategies on a parse tree example taken from our training corpus. The parse tree in the figure is relatively small, and doesn't leave so much choice on the assignment of head annotation. Nevertheless, it is possible to understand the main features of elementary trees extracted using the different strategies. In particular we can see how the *EM-LTSG* chooses the longest possible fragment rooted in the TOP node of the parse tree.

4.2 Growth of the grammars

4.2.1 Coherence

We can try to estimate the quality of the one-anchored LTSGs by analyzing their growth during the training phase. The "ideal" grammar with the largest *generalization power*, should stop to grow after enough sentences are provided as input. In this case, we would in fact expect a decrease in the number of new constructions as the grammar grows in size.
It's important at this point to distinguish between the *generalization power* of a grammar and its *coherence*: the first refers to the ability to learn fragments which are general enough to occur in different sentences, while the second refers to the ability of extracting elementary trees which are more probable to be later extracted in new parse trees with the same strategy. In our analysis we are only able to measure the coherence of a grammar, which is not always a valid indicator of its generalization power. A grammar could in fact discover a novel fragment from an input parse tree, while already containing all the elementary trees necessary to parse it.

4.2.2 New fragments

Figure 4.2 shows, for each different strategy, the growth of the number of new fragments as the training corpus grows in size. No matter which formalism is chosen, a large number of fragments remains left out from the grammar, since the amount of new fragments which continue to appear is stationary as the training data grows.The thick line in the figure, referring to the grammar constituted by all possible elementary trees, represents the general trend common to the various strategies. Although all the strategies have this trend in common, we can observe remarkable differences. In particular we can notice a sharp separations between the "upper" strategies (*EM-LTSG*, *Entropy-LTSG*, *Random-LTSG*, *Right-LTSG*) whose grammar growth is stationary at around 1.5 new fragments for every new input tree, and the "lower" strategies (*Collins-LTSG*, *Left-LTSG*, *Greedy-LTSG*) whose growth is stationary at around 0.5 new fragments for every new input tree. A similar analysis can be done observing the graph of Figure 4.3 showing a similar separation between the strategies when counting the number of elementary trees occurring 1, 2, 3 or 4 times in the grammars.

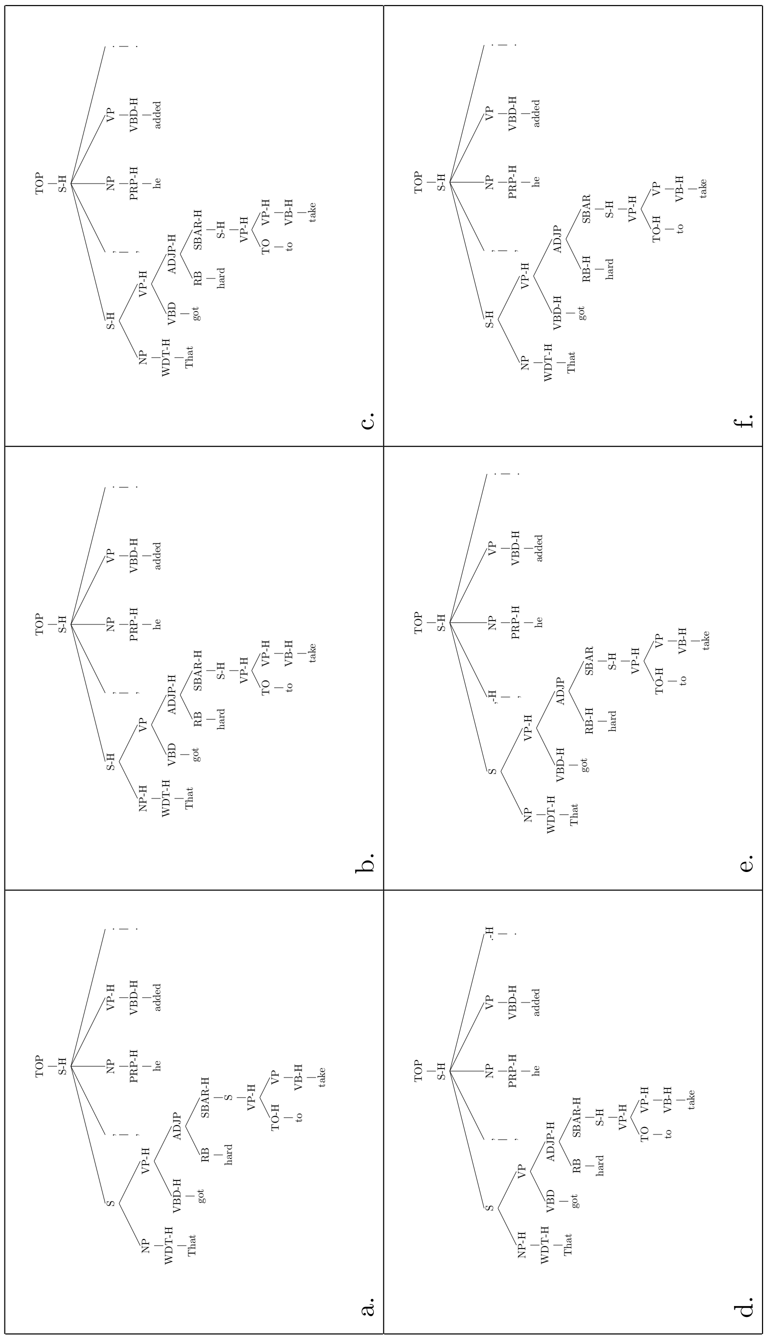


Figure 4.1: Example of head annotation using a. Collins and Magerman scheme, b. one of the possible random assignment, c. EM-1best algorithm, d. Entropy algorithm, e. Greedy (Top) heuristic, f. Greedy (Bottom) heuristic.

Given this separation, we can reasonably say that the "lower" strategies have higher coherence. From the results shown in Table 3.2, we know that two of the "upper" strategies, namely *Random-LTSG* and *Entropy-LTSG*, have a very good performance. On the other hand the *Collins-LTSG* and *Left-LTSG* annotations which seem to generalize quite well do not perform as good as other strategies. If we assume that the performance of a grammar is a valid indicator of its generalization power, we can conclude that *Random-LTSG* and *Entropy-LTSG* have good generalization power and low coherence while *Left-LTSG* and *Collins-LTSG* are coherent but worse in the generalization ability. Finally *EM-LTSG* is both incoherent and poor in the generalization, for the overfitting phenomenon discussed in section 3.4.3, and *Greedy-LTSG* is the most coherent grammar and it is aligned with the best performance strategies.

4.2.3 The random strategy

The random strategy is a good example to explain why coherence is not always a valid indicator of good performance. It is easy to understand where the incoherence of the model lies: this strategy finds a huge number of new constructions because it follows a random annotation of the parse tree.
We would also like to have a better understanding of why this model performs so well. According to our intuition this is due to the fact that this strategy learns a big number of constructions, not too small nor too big in size, which turns out to be general enough when parsing new sentences. If we have a closer look at the elementary trees extracted in this grammar (Appendix B), we realize in fact that they are not as short as the *Left-LTSG* but also do not fall in the overfitting problem encountered in *EM-LTSG*, characterized by very large elementary trees.
This can offer a valid explanation if we consider the general skewness in the structure of linguistic sentences and the process of randomly annotating a parse tree. We know in fact that linguistic structure usually presents a certain degree of skewness [Seginer, 2007]. In simple terms this basically means that for every internal node A of a parse tree, there is usually an uneven distribution of the mass of the subtree rooted in A under each single daughter of A. For this reason, when attributing random head dependencies, the probability of constantly assigning the head mark to the "heaviest" or to the "lightest" daughter is considerably small.

4.3 Qualitative analysis

In order to have a better understanding of the type of head assignment given by each single strategy, we will try to analyze the nature of the new fragments which continue to appear as the grammar is being trained. In particular we are interested to understand if, when encountering a novel elementary tree, there are other "similar" constructions present in the grammar. This idea of similar-

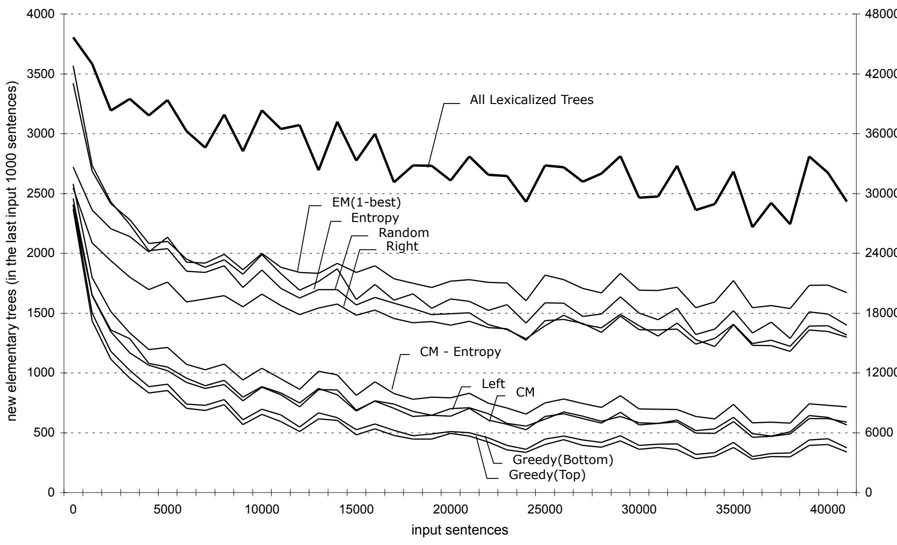


Figure 4.2: Growth of new elementary trees every new 1000 sentences in WSJ40.

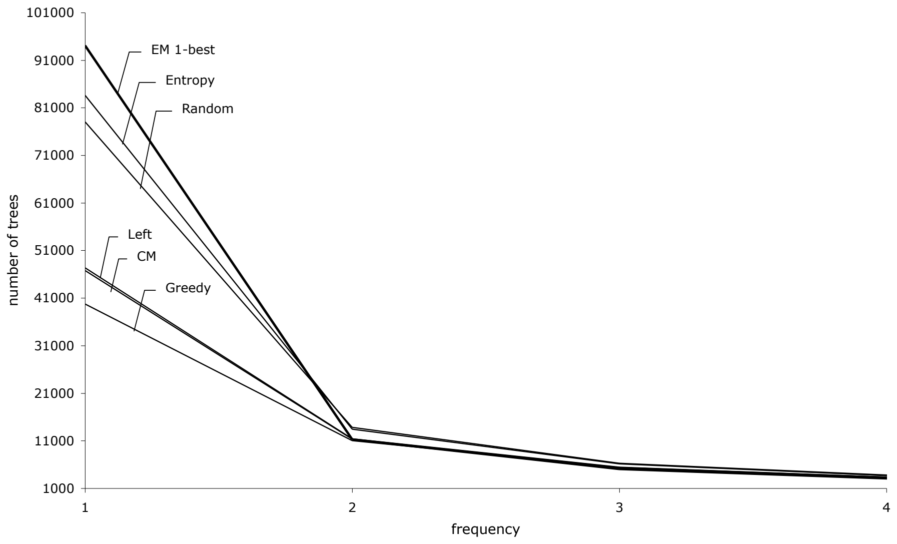
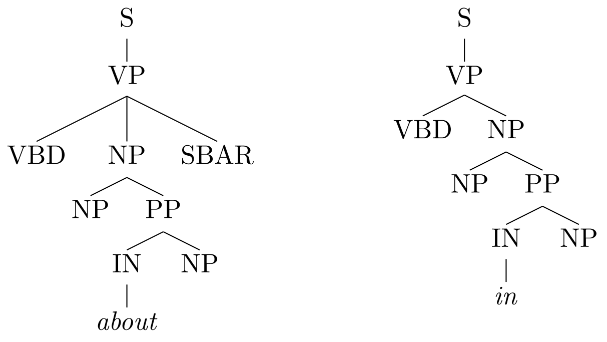


Figure 4.3: Number of trees occurring 1 to 4 times in different one-anchored LTSG.

ity is in a way a possible extension to the notion of coherence in a grammar. We would in fact prefer a coherent grammar to extract constructions which are possibly identical, or at least similar to the ones already present in the grammar.

4.3.1 Similarity measure between fragments

The notion of similarity that we are going to define is based on the concept of comparability. Specifically two trees are similar only if they are comparable. We consider two elementary trees to be comparable when they share the same lexical path (or spine). If we remember the way we defined the notion of lexical path in section 2.4.2, we understand that two elementary trees are comparable if they have the same depth and if they share the same labeling of the head-annotated internal nodes. This categorization works for both lexicalized or delexicalized trees[1]. As an example the following elementary trees, all share the same lexical path:



In order to evaluate a measure of similarity between two elementary trees sharing the same lexical path, we compare their internal nodes. Specifically we sum for each level of depth the number of shared elements. This number divided by the total number of internal nodes of the first and the second tree, would then account for some notion of recall and precision respectively. We will take the harmonic mean between recall and precision to account for the measure of similarity.
We explicitly decided to ignore the ordering of the internal nodes when processing this measurements. We consider in fact two constructions to have a maximum degree of similarity when they have the same internal nodes at corresponding depth, even if not in the same order.

[1]The delexicalization of the tree fragments generates tree *templates* similar to the ones described in [Chiang, 2003].

4.3.2 Transformation rules

The notion of similarity between elementary trees, is useful when we we want to keep track of new elementary trees appearing in our grammar during the learning phase. Every time that a fragment appears for the first time, we would like to see which are the most similar trees in our grammar.
The appendices at the end of this thesis report a sample of the new elementary trees encountered at the end of the training phase when running the main strategies analyzed so far. Each table presents a newly encountered elementary tree (left) together with few samples of fragments (right) present in the grammar which maximize the similarity measure with that tree.
We believe that this kind of analysis could give interesting insight on possible directions we can take to improve the quality of our grammars. It's beyond the scope of this thesis to undergo any such direction. We will nevertheless try to identify a few interesting features which occur in those fragments. The idea is to come up with a limited number of *transformation rules* which could enable our grammar to generalize new constructions.
After analyzing the kind of trees reported in the appendixes, we come up with 4 simple transformation rules that could allow many newly encountered tree to be converted to one of the existing trees in the grammar. Among the reported example there are cases in which neither one of these 4 transformation rules could intuitively apply; some of these are cases of punctuation marks singularity (as in the example at page 78) and manual annotation mistakes (as in the examples at page 66 and 75).
The 4 types of transformation rules are:

1. **Displacement**: there is a permutated version of the new tree in the grammar.
   - ADVP (before or after the verb) as on page 74
   - PRT (right after the verb or after its object) as on page 76
2. **Substitution**: a category can be replaced with an other category.
   - VBD $\leftrightarrow$ VBP (present - past form) as on page 82(bottom)
   - S $\leftrightarrow$ SBAR (simple declarative - subordinate clause) as on page 83
3. **Multiplication - Reduction**: a single category is replaced by two instances of the same category or the other way around.
   - NN* (multiples nouns in compound nouns) as on page 71
   - CD* ("millions" can optionally follow a number) as on page 72
   - JJ* (multiple consecutive adjectives) as on page 82(top)
4. **Deletion - Insertion**: a single category can be omitted or inserted.
   - PRT (after the verb "points out" - "says") as on page 68

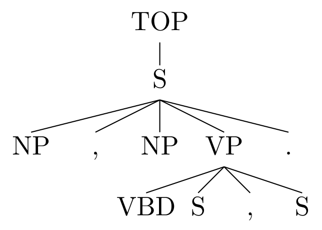
- PP (prepositional phrase after the verb) as on page 67
- RB (negation before the verb) as on page 70
- CD (number insertion between DT and NNS) as on page 79
- DT (determiners can be omitted before NNP) as on page 80
- ADVP (optional “so” at the beginning of a sentence) as on page 84

4.4 Conclusions

In this thesis we have started to examine the general class of Tree Substitution Grammars. Among its possible implementations, DOP is the approach which leads to the best results, and in fact the only one in this family within the state-of-the-art methodologies. Nevertheless we were faced with the problem of its complexity which didn’t allow us to use it for parsing reasonably long sentences. Among the other TSGs, we have studied the subclass of LTSGs generated by head-dependency structures. We were inspired by the simplicity of its constraints and impressed by the encouraging results both in terms of efficiency and performance.
Furthermore we have discovered a number of heuristic based strategies which could improve the current state-of-the-art head dependency scheme. We are aware that these improvements are modest and that our new scores are still far from the best scores in the field. Nevertheless it is important to mention that the kind of fragments that we are extracting are used in more sophisticate grammars. For instance Collins and Magerman’s head annotation scheme is used in successful parsers adopting tree adjoining grammars [Chiang, 2003].
We therefore reserve some optimism in possible further refinement of our methodology. In particular, after analyzing the nature of the elementary trees extracted from our grammars, we have come to realize that when they fail to fit novel encountered constructions, the differences are based on regular schemes. For this reason we have sketched few examples of simple *transformation rules* which could allow to relate novel fragments with existing ones. Although these techniques are not based on solid evidence, we are convinced that further investigations in this direction are worthy to pursue.

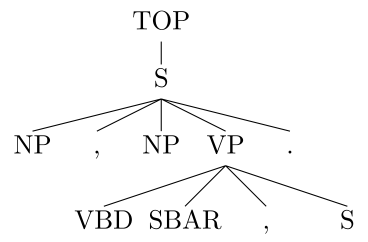
Appendix A

CM-LTSG

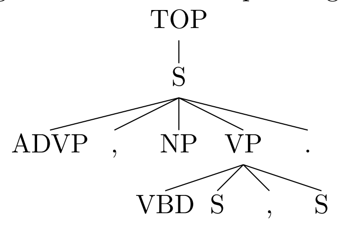


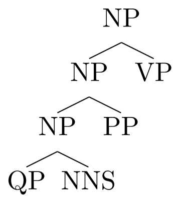
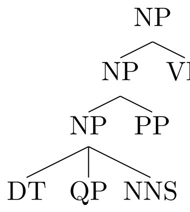
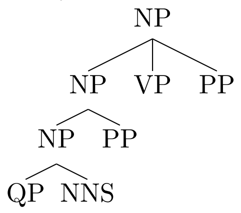
Yesterday , Mr. Trump tried to put the blame for the collapse of the UAL deal on Congress , saying it was rushing through a bill to protect AMR executives .

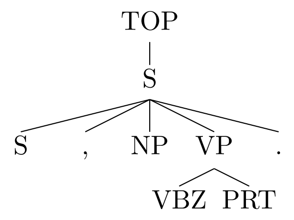
Last year , a federal appeals court in St. Louis said the preamble was unconstitutional , citing an earlier Supreme Court ruling that states ca n't justify stricter abortion curbs by changing the definition of when life begins .



A few months ago , the Bush administration decided to stop this cooperation , leaving Radio Costa Rica operating on a shoestring .

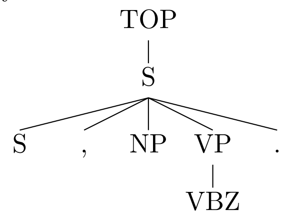


 There are some important caveats : Before investing in stocks , individuals should have at least three to six months of living expenses set aside in the bank , most investment advisers say .	Still , flat-rolled is the steel industry 's bread and butter , representing about half of the 80 million tons of steel expected to be shipped this year . 
	However , the French merchant has about 200,000 tons of old crop Ivory Coast cocoa stored in the Netherlands from an agreement it had negotiated with the Ivory Coast last spring . 

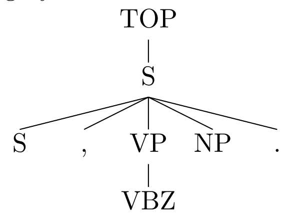


That cuts the risk , Mr. Gregory , the San Francisco money manager , points out .

That 's not all , he says .

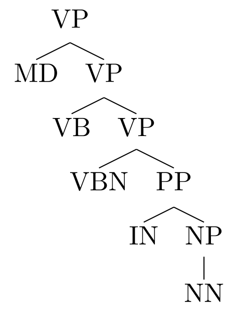
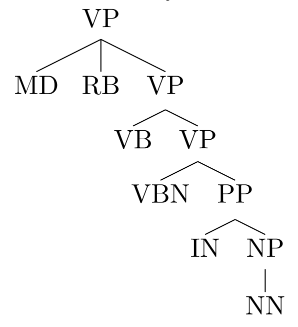


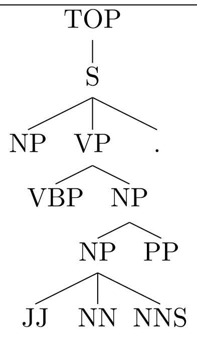
Still others are stung by a desire to do both well and good , says Douglas Watson , commanding officer of the Los Angeles Police Department 's bunko-forgery division .



Appendix B

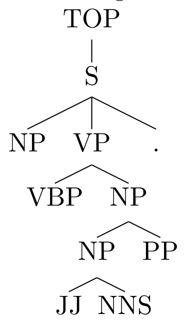
Random-LTSG

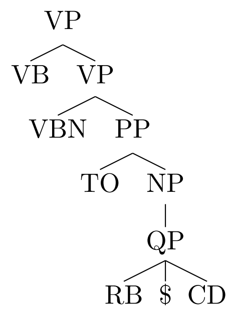
 Neither they nor Mr. McAlpine could be reached for comment .	Mr. Cray , who could n't be reached for comment , will work for the new Colorado Springs , Colo. , company as an independent contractor – the arrangement he had with Cray Research . 



Individual investors face high transaction costs of moving in and out of the market .

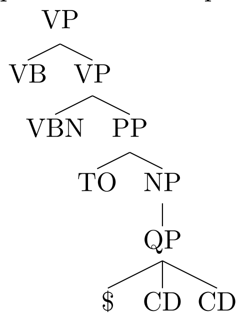
Most earn high ratings from credit agencies .



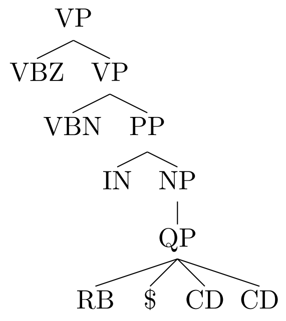


But a dollar invested in long-term bonds in 1926 would have grown to only $ 16.56 , and a dollar put in Treasury bills would equal a meager $ 9.29 .

The August gap was expected to have expanded to $ 9.1 billion .

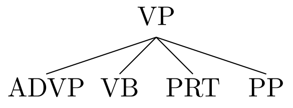
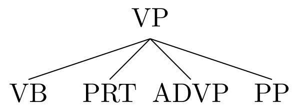
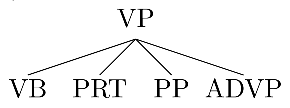
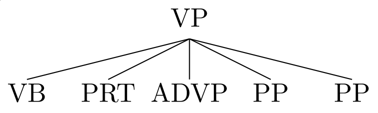
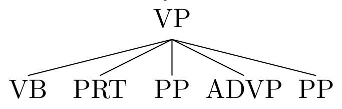


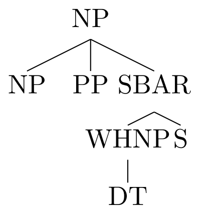
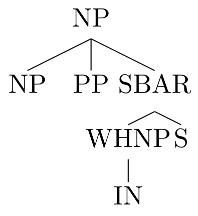
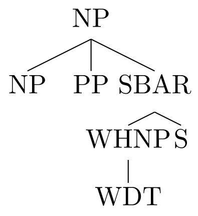
The new company is capitalized at about $ 3.5 million .

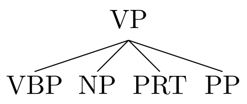
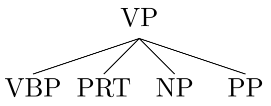
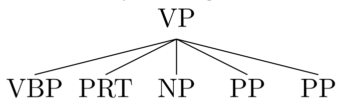
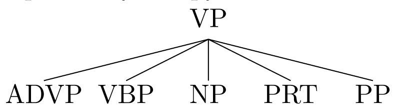
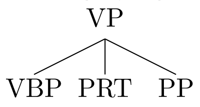
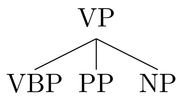


Appendix C

Greedy(Bottom)-LTSG

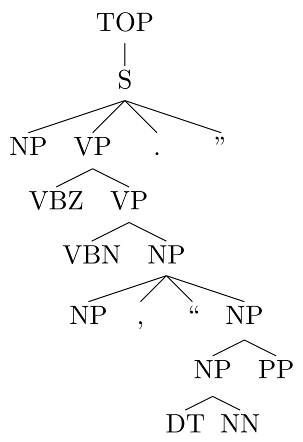
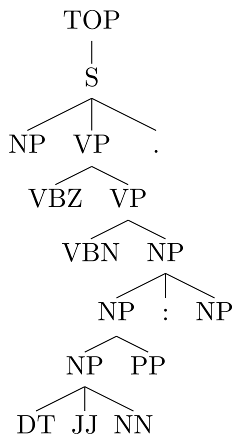
 Why does n't he just follow through on one of these things ? "	Spooked investors , despite their stampede to dump takeover stocks , should hold on tight to their Jaguar shares . 
	Last year 's drought in the Midwest prompted retailers to stock up on oils ahead of anticipated price increases , boosting sales for Crisco and Puritan oils , analysts said . 
	The counter-argument , which he has heard , is that if he and his fellow Democrats are successful in killing the president 's proposal , the revenue gap will open up tremendously in 1990 because of the weakened economy . 
	AN AUSTIN , Texas , company plans to make it easy for you show up in court a thousand miles away without leaving town . 

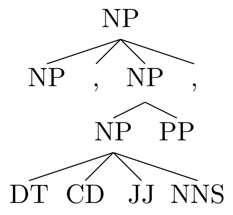
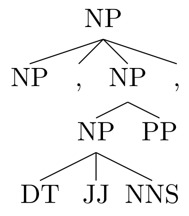
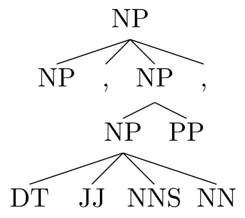
 -LRB- During its centennial year , The Wall Street Journal will report events of the past century that stand as milestones of American business history . -RRB-	Assuming it was n't one of those columns that you clipped and put on the refrigerator door , I 'll review the facts . 
	Economists say a buildup in inventories can provoke cutbacks in production that can lead to a recession . 

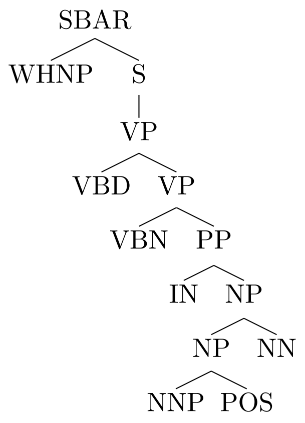
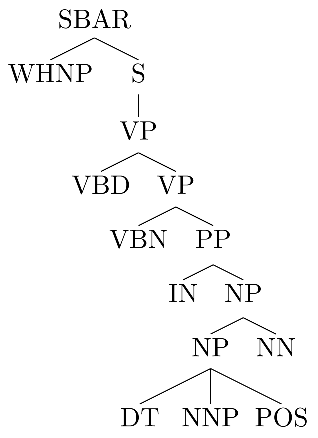
 Otherwise , if you put all your money in at one time , by sheer bad luck , you might pick a terrible time , and have to wait three years to get even , Mr. Gregory says .	Instead , they map out a strategy in several phases from now until 1995 . 
	Beginning as a laborer in a refinery , Mr. Guzman Cabrera put in more than 40 years at Pemex before being pushed into retirement by La Quina after a dispute two years ago . 
	I have a right to print those scripts if I go there and laboriously – but no longer surreptitiously – copy them out in long hand . 
	HEALTH CLUBS gear up for a graying clientele . 
	I agree with Mr. Lehman 100 % ! 

Appendix D

EM(1-best)-LTSG

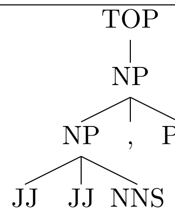
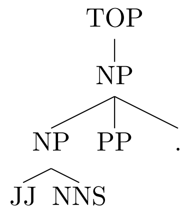
 TOP S NP VP . ” VBZ VP VBN NP NP , “ NP NP PP DT NN He ’s written this book , ‘ The Art of the Deal . ’	Japan has found another safe outlet for its money : U.S. home mortgages .  TOP S NP VP . VBZ VP VBN NP NP : NP NP PP DT JJ NN

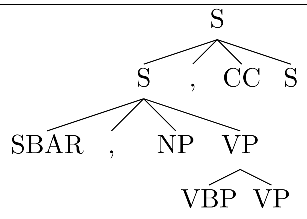
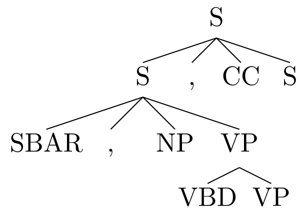
 Meanwhile , Citicorp and Chase Manhattan Corp. , the two lead lenders on the UAL buy-out , met with other banks yesterday to determine if they would be willing to finance the buy-out at a lower price .	Two of the major factors buoying prices , the prolonged strikes at the Highland Valley mine in Canada and the Cananea mine in Mexico , were finally resolved . 
	William Tait , a former sales manager in Indianapolis , says that his office had all but sold a $ 1.5 million image system to pharmaceutical maker Eli Lilly & Co . 

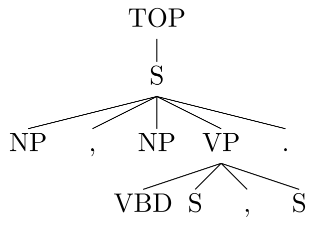
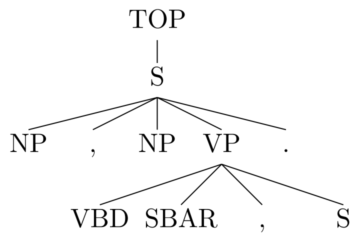
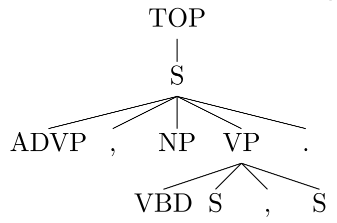
 Los Angeles investor Marvin Davis , whose $ 275-a-share offer was rejected by UAL 's board , has n't shown signs of pursuing a $ 300-a-share back-up bid he made last month .	Fed Vice Chairman Manuel Johnson , who had dissented from the Treasury 's policy , told lawmakers , “ I became convinced about what looked to me like an attempt to push the dollar down against the fundamentals in the market . ” 

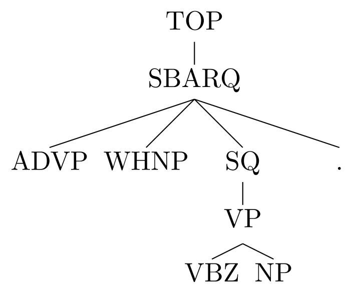
Appendix E

Entropy-LTSG

 Compound annual returns , including price changes and income from interest and dividends	Indentical conditions for the two parts . 

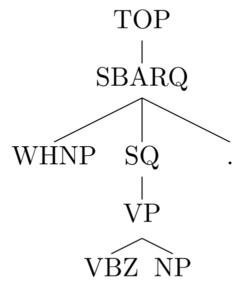
 “ When the market is low , you are buying more shares , and when it ’s high , you ’re buying fewer shares , ” he says .	Once the disease was confirmed , all the man ’s associates and family were tested , but none have so far been found to have AIDS , the newspaper said . 

 Yesterday , Mr. Trump tried to put the blame for the collapse of the UAL deal on Congress , saying it was rushing through a bill to protect AMR executives .	Last year , a federal appeals court in St. Louis said the preamble was unconstitutional , citing an earlier Supreme Court ruling that states ca n't justify stricter abortion curbs by changing the definition of when life begins .  TOP S NP , NP VP . VBD SBAR , S
	A few months ago , the Bush administration decided to stop this cooperation , leaving Radio Costa Rica operating on a shoestring .  TOP S ADVP , NP VP . VBD S , S

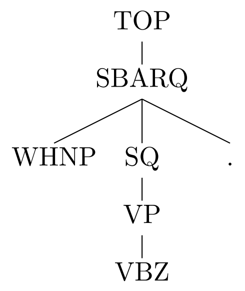


So what 's the best way to buy stocks ?

WHICH IS the best medicine for runaway health costs : competition or regulation ?



Who cares ?



Bibliography

Steven Abney. Statistical methods and linguistics. In Judith Klavans and Philip Resnik, editors, *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*, pages 1–26. The MIT Press, Cambridge, Massachusetts, 1996. URL `citeseer.ist.psu.edu/abney96statistical.html`.

Rens Bod. What is the minimal set of fragments that achieves maximal parse accuracy? In *Proceedings ACL-2001*. Morgan Kaufmann, San Francisco, CA, 2001.

Rens Bod. *Beyond Grammar: An Experience-Based Theory of Language*. CSLI Publications, Cambridge University Press, 1998.

Rens Bod. A computational model of language performance: Data oriented parsing. In *Proceedings COLING'92 (Nantes, France)*, pages 855–859. Association for Computational Linguistics, Morristown, NJ, 1992.

Rens Bod. Using an annotated corpus as a stochastic grammar. In *Proceedings of the sixth conference on European chapter of the Association for Computational Linguistics*, pages 37–44, Morristown, NJ, USA, 1993. Association for Computational Linguistics. ISBN 90-5434-014-2. doi: http://dx.doi.org/10.3115/976744.976750.

Rens Bod, Remko Scha, and Khalil Sima'an, editors. *Data-Oriented Parsing*. CSLI Publications, University of Chicago Press, Chicago, IL, 2003.

Remko Bonnema and Remko Scha. Reconsidering the probability model for dop. In Bod et al. [2003], pages 25–41.

Remko Bonnema, Paul Buying, and Remko Scha. A new probability model for data oriented parsing. In Paul Dekker, editor, *Proceedings of the Twelfth Amsterdam Colloquium*. ILLC, University of Amsterdam, 1999.

Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the fourteenth national conference on artificial intelligence*, Menlo Park, 1997. AAAI Press/MIT Press.

D. Chiang and D. Bikel. Recovering latent information in treebanks, 2002. URL `citeseer.ist.psu.edu/article/chiang02recovering.html`.

David Chiang. Statistical parsing with an automatically extracted tree adjoining grammar. In Bod et al. [2003].

N. Chomsky. Three models for the description of language. *Information Theory, IEEE Transactions on*, 2(3):113–124, 1956.

Noam Chomsky. *Aspects of the theory of syntax.* MIT Press, Cambridge, MA, 1965.

Michael Collins. Three generative, lexicalized models for statistical parsing. In Philip R. Cohen and Wolfgang Wahlster, editors, *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics*, pages 16–23. Association for Computational Linguistics, Somerset, NJ, 1997. URL `citeseer.nj.nec.com/collins97three.html`.

Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

Joshua Goodman. Efficient algorithms for parsing the DOP model. In Eric Brill and Kenneth Church, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 143–152. Association for Computational Linguistics, Somerset, New Jersey, 1996. URL `citeseer.ist.psu.edu/article/goodman96efficient.html`.

Joshua Goodman. Efficient parsing of dop with pcfg-reductions. In Bod et al. [2003], pages 191–210.

A. K. Joshi. How much context-sensitivity is required to provide reasonable structural descriptions: Tree-adjoining grammars. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Parsing: Psycholinguistic, Computational and Theoretical Perspectives*, pages 206–350. Cambridge University Press, New York, 1985.

Aravind K. Joshi. Starting with complex primitives pays off: complicate locally, simplify globally. *Cognitive Science*, 28(5):637–668, 2004.

Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–124. Springer, Berlin, New York, 1997. URL `citeseer.ist.psu.edu/joshi97treeadjoining.html`.

T. Kasami and K. Torii. A syntax-analysis procedure for unambiguous context-free grammars. *J. ACM*, 16(3):423–431, 1969. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/321526.321531.

David M. Magerman. Statistical decision-tree models for parsing. In *Meeting of the Association for Computational Linguistics*, pages 276–283, 1995. URL `citeseer.ist.psu.edu/magerman95statistical.html`.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1994. URL `citeseer.ist.psu.edu/marcus04building.html`.

William A. Martin, Kenneth W. Church, and Ramesh S. Patil. Preliminary analysis of a breadth-first parsing algorithm: theoretical and experimental results. pages 267–328, 1987.

Detlef Prescher. *EM-basierte maschinelle Lernverfahren fr natrliche Sprachen*. PhD thesis, Universitt Stuttgart, Institut fr Maschinelle Sprachverarbeitung (IMS), Stuttgart, 2002. URL `Prescher:2002:EBM.pdf`.

Detlef Prescher. A tutorial on the expectation-maximization algorithm including maximum-likelihood estimation and em training of probabilistic context-free grammars, March 2005. URL `http://arxiv.org/abs/cs/0412015`.

Remko Scha. Taaltheorie en taaltechnologie; competence en performance. In R. de Kort and G.L.J. Leerdam, editors, *Computertoepassingen in de Neerlandistiek*, pages 7–22. LVVN, Almere, the Netherlands, 1990. English translation at http://iaaa.nl/rs/LeerdamE.html.

Helmut Schmid. Efficient parsing of highly ambiguous context-free grammars with bit vectors. In *COLING '04: Proceedings of the 20th international conference on Computational Linguistics*, page 162, Morristown, NJ, USA, 2004. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1220355.1220379.

Yoav Seginer. *Learning Syntactic Structure*. PhD thesis, University of Amsterdam, 2007.

Satoshi Sekine and Michael John Collins. evalb. http://cs.nyu.edu/cs/projects/proteus/evalb.

Stuart M. Shieber. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343, 1985.

Khalil Sima'an. Computational complexity of probabilistic disambiguation by means of tree grammars. In *Proceedings of COLING'96*, volume 2, pages 1175–1180, Copenhagen (Denmark), 1996. URL `citeseer.ist.psu.edu/article/simaan96computational.html`. cmp-lg/9606019.

A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Information Theory*, IT-13:260–269, 1967.

Andreas Zollmann and Khalil Sima'an. A consistent and efficient estimator for data-oriented parsing. *Journal of Automata, Languages and Combinatorics*, 10(2/3):367–388, 2005.