STATISTICAL PARSING ALGORITHMS FOR LEXICALIZED TREE ADJOINING GRAMMARS

Anoop Sarkar

A DISSERTATION PROPOSAL

in

Computer and Information Science

Abstract

STATISTICAL PARSING ALGORITHMS FOR LEXICALIZED TREE ADJOINING GRAMMARS

Anoop Sarkar

Supervisor: Professor Aravind Joshi

The goal of this dissertation is two-fold: to develop the theory of probabilistic Tree Adjoining Grammars (TAGs) and to present some practical results in the form of efficient parsing and estimation algorithms for probabilistic TAGs.

The overall goal of developing the theory of probabilistic TAGs is to provide a simple, mathematically and linguistically well-formed probabilistic framework for statistical parsing.

The practical results in parsing and estimation of probabilistic TAGs are developed with a view towards an increasingly unsupervised approach to the training of statistical parsers and language models.

In particular, this proposal contains the following results:

- An algorithm for determining deficiency in a generative model for probabilistic TAGs.
- A novel chart based head-corner parsing algorithm for probabilistic TAGs.
- A probability model for statistical parsing and a co-training method for training this parser which combines labeled and unlabeled data.
- An algorithm for computing prefix probabilities which can be used to predict the word most likely to occur after an initial substring of the input.

The proposed work can be summarized in the following points:

- A separate evaluation of the co-training algorithm on a larger set of labeled and unlabeled data, in addition to the evaluation presented in this proposal.

- An evaluation of the prefix probability algorithm by comparing it with a trigram language model.

- An extension of techniques in learning subcategorization information and verb classes to produce TAG lexicons which can be directly used to improve performance of the co-training algorithm.

Contents

Chapter 1

Introduction

This dissertation proposal introduces algorithms for statistical parsing and prefix probability computation using probabilistic Tree Adjoining Grammars (TAGs).

Probabilistic TAGs are a model for natural language syntax which permits the local definition of predicate-argument structure. Such a representation allows independence assumptions in a statistical parser to be linked naturally to the notion of locality. As a result of this representation, complex syntactic phenomena are given a simple probability model in which consistency is easily defined. Another consequence of a fully lexicalized structural description is that recursive phenomena of language can be treated as a classification problem of the form commonly addressed in the machine learning literature. This allows the use of increasingly unsupervised methods in natural language processing by exploiting the combination of labeled and unlabeled training data. Finally, the parsing of strings using probabilistic TAGs can be generalized to the parsing of initial substrings allowing its application to language modeling.

This dissertation provides several concrete results in the theory of probabilistic TAGs. The overall goal of developing this theory is to provide a simple, mathematically and linguistically well-formed probabilistic framework for topics in statistical parsing. We also give practical results in parsing and estimation of probabilistic TAGs with a view towards an increasingly unsupervised approach to the training of statistical parsers and language models.

1.1 Outline of the Dissertation

This dissertation proposal presents several results that are related to statistical parsing using TAGs. The various results that relate to this overall theme are organized into the following parts in this proposal.

1. Chapter 2 gives a definition of probabilistic measures applied to TAGs. This chapter contains an algorithm for determining deficiency in a generative model when TAG derivations are assigned probabilities. This provides a proper and consistent generative model for statistical parsing.

2. Chapter 3 defines a chart based head-corner parsing algorithm for probabilistic TAGs. This parsing algorithm takes as input a probabilistic TAG which is estimated using techniques described in Chapter 4.

3. Chapter 4 presents a probability model for statistical parsing using TAGs and a Co-training method for training this parser by combining labeled and unlabeled data.

4. Chapter 5 contains an algorithm for computing prefix probabilities which can be used to predict the word most likely to occur after having seen an initial substring of the input. This generalizes the parsing algorithm given in Chapter 3 which deals with entire sentences to deal with initial substrings of sentences.

1.2 Reading Guide

For those readers familiar with probabilistic TAGs, to get a quick grasp of the contents of the dissertation proposal, you can read through the following sections (a total of about 20 pages):

- Section 1.1 for an outline of thesis and the underlying theme behind the various results.
- Section 3.1 and Section 3.2 for a brief introduction to the TAG parsing algorithm introduced in this dissertation.

- Section 4.1 and Section 4.2 for an introduction to the use of co-training methods which exploits the representation oflexicalized grammars.
- Section 5.3 gives an overview of the algorithm for prefix probability computation.

For some further details on parts not covered in the sections mentioned above, the reader can read the following sections (for a total of about 50 pages):

- Section 2.3 and Section 2.4 for the definition of consistency in a generative probabilistic TAG model.
- Section 4.3 and Section 4.4 for more details on the estimation of probabilistic TAGs using co-training.

For those unfamiliar to probabilistic TAGs, an introduction to TAG is given in Section 1.3 and an introduction to probabilistic TAGs is in Section 1.5.

1.3 Tree Adjoining Grammars

Probabilistic Tree Adjoining Grammars (TAGs) are a model for natural language syntax which allows the local definition of predicate-argument structure represented as lexicalized elementary trees. An important aspect of statistical parsing are the independence assumptions made in the probability models. A TAG representation allows independence assumptions in a statistical parser to be linked naturally to the notion of locality.

We present in this thesis algorithms for statistical parsing and prefix probability computation using probabilistic TAGs. In particular, this thesis contains the following results:

(a) The definition of probabilistic measures applied to TAGs. We give an algorithm for determining deficiency in a generative model when TAG derivations are assigned probabilities.

(b) A chart based head-corner parsing algorithm for probabilistic TAGs.

(c) A probability model for statistical parsing using TAGs and a Co-training method for training this parser by combining labeled and unlabeled data.

(d) An algorithm for computing prefix probabilities which can be used to predict the word most likely to occur after an initial substring of the input.

We also discuss methods for lexical knowledge acquisition from corpora that can be used for unsupervised learning of parsers for natural language that exploit the locality of predicate-argument relations in a probabilistic TAG.

1.4 Lexicalized Tree Adjoining Grammars

We give a brief overview of the Tree Adjoining Grammar (TAG) formalism [Joshi et al., 1975], extended to include lexicalization [Schabes et al., 1988]. Tree Adjoining Languages (TALs) fall into the class of mildly context-sensitive languages, and as such are more powerful than context free languages. The TAG formalism in general, and lexicalized TAGs in particular, are well-suited for linguistic applications. As first shown by [Joshi, 1985] and [Kroch and Joshi, 1987], the properties of TAGs permit us to encapsulate diverse syntactic phenomena in a very natural way. For example, TAG's extended domain of locality and its factoring of recursion from local dependencies lead, among other things, to a localization of so-called unbounded dependencies.

1.4.1 TAG formalism

The primitive elements of the standard TAG formalism are known as elementary trees. ELEMENTARY TREES are of two types: initial trees and auxiliary trees (see Figure 1.1). In describing natural language, INITIAL TREES are minimal linguistic structures that contain no recursion, i.e. trees containing the phrasal structure of simple sentences, NP's, PP's, and so forth. Initial trees are characterized by the following: 1) all internal nodes are labeled by non-terminals, 2) all leaf nodes are labeled by terminals, or by non-terminal nodes marked for substitution. An initial tree is called an X-type initial tree if its root is labeled with type X.

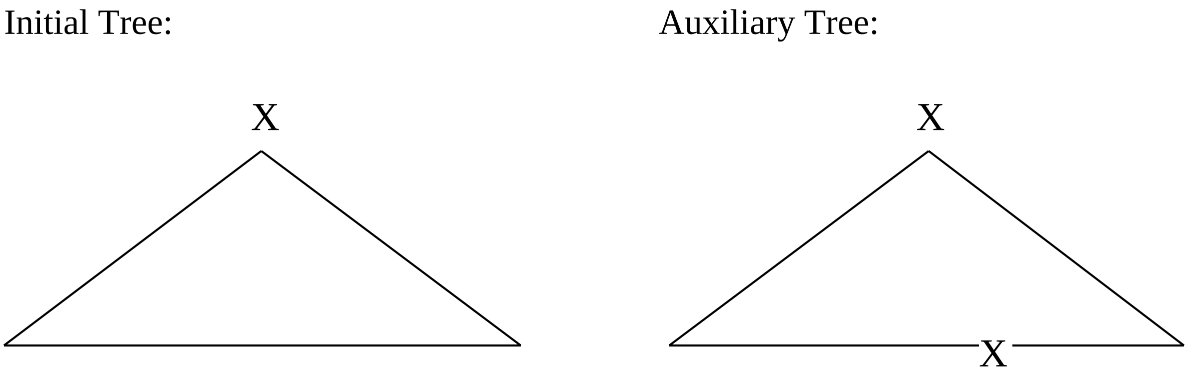


Figure 1.1: Elementary trees in TAG

Recursive structures are represented by AUXILIARY TREES, which represent constituents that are adjuncts to basic structures (e.g. adverbials). Auxiliary trees are characterized as follows: 1) all internal nodes are labeled by non-terminals, 2) all leaf nodes are labeled by terminals, or by non-terminal nodes marked for substitution, except for exactly one non-terminal node, called the foot node, which can only be used to adjoin the tree to another node[1], 3) the foot node has the same label as the root node of the tree.

There are two operations defined in the TAG formalism, substitution[2] and adjunction. In the SUBSTITUTION operation, the root node on an initial tree is merged into a non-terminal leaf node marked for substitution in another initial tree, producing a new tree. The root node and the substitution node must have the same name. Figure 1.2 shows two initial trees and the tree resulting from the substitution of one tree into the other.

In an ADJUNCTION operation, an auxiliary tree is grafted onto a non-terminal node anywhere in an initial tree. The root and foot nodes of the auxiliary tree must match the node at which the auxiliary tree adjoins. Figure 1.3 shows an auxiliary tree and

[1] A null adjunction constraint (NA) is systematically put on the foot node of an auxiliary tree. This disallows adjunction of a tree onto the foot node itself.

[2] Technically, substitution is a specialized version of adjunction, but it is useful to make a distinction between the two.

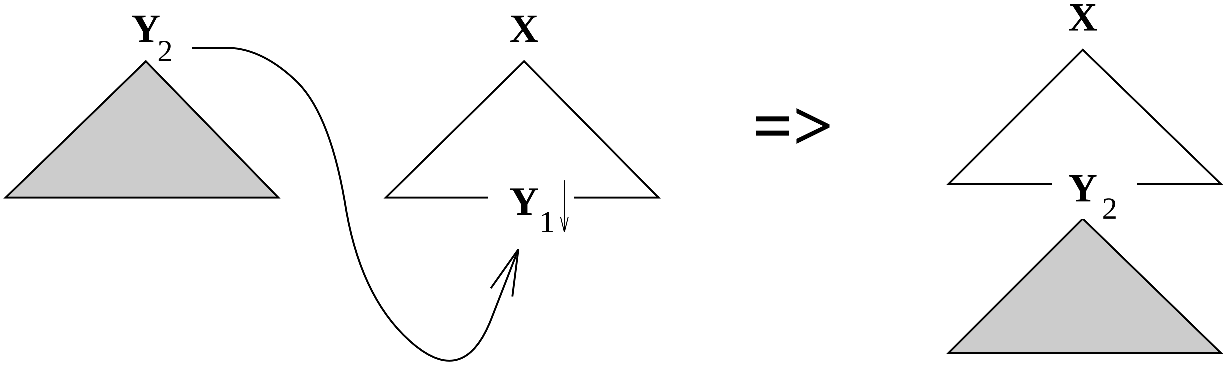


Figure 1.2: Substitution in TAG

an initial tree, and the tree resulting from an adjunction operation.

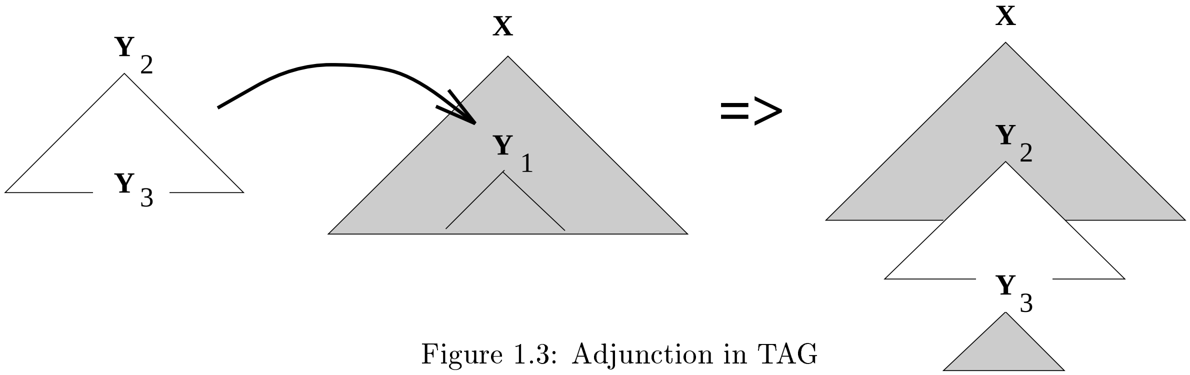


Figure 1.3: Adjunction in TAG

A TAG G is a collection of finite initial trees, I, and auxiliary trees, A. The TREE SET of a TAG G, $\mathcal{T}(G)$ is defined to be the set of all derived trees starting from S-type initial trees in I whose frontier consists of terminal nodes (all substitution nodes having been filled). The STRING LANGUAGE generated by a TAG, $\mathcal{L}(G)$, is defined to be the set of all terminal strings on the frontier of the trees in $\mathcal{T}(G)$.

1.4.2 Lexicalization

'Lexicalized' grammars systematically associate each elementary structure with a lexical anchor. This means that in each structure there is a lexical item that is realized. It does not mean simply adding feature structures (such as head) and unification equations to the rules of the formalism. These resultant elementary structures specify extended domains of locality (as compared to CFGs) over which constraints can be stated.

Following [Schabes et al., 1988] we say that a grammar is LEXICALIZED if it consists of 1) a finite set of structures each associated with a lexical item, and 2) an operation or operations for composing the structures. Each lexical item will be called the ANCHOR of the corresponding structure, which defines the domain of locality over which constraints are specified. Note then, that constraints are local with respect to their anchor.

Not every grammar is in a lexicalized form.[3] In the process of lexicalizing a grammar, the lexicalized grammar is required to be strongly equivalent to the original grammar, i.e. it must produce not only the same language, but the same structures or tree set as well.

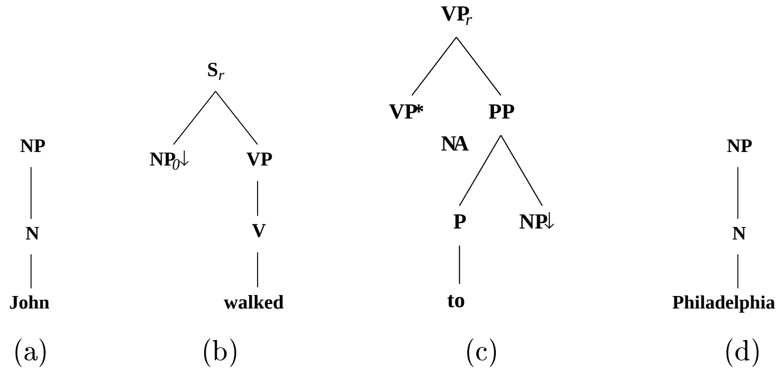


Figure 1.4: Lexicalized Elementary trees

In Figure 1.4, which shows sample initial and auxiliary trees, substitution sites are marked by a $\downarrow$, and foot nodes are marked by an $*$. This notation is standard and is followed in the rest of this thesis unless noted otherwise. In this introduction chapter we will occasionally make the terms foot and substitution nodes explicit to make it more recognizable for the reader.

[3] Notice the similarity of the definition of a lexicalized grammar with the off line parsability constraint ([Kaplan and Bresnan, 1983]). As consequences of our definition, each structure has at least one lexical item (its anchor) attached to it and all sentences are finitely ambiguous.

1.5 Probabilistic Tree Adjoining Grammars

We explain the basic model for probabilistic parsing using an LTAG by using an example. Consider an example sentence: *Ms. Haag plays Elianti.* In an LTAG, each word in this sentence is associated with a set of *elementary trees.* There are two kinds of elementary trees. Certain trees with distinguished non-terminal node (termed as footnode) in the frontier of the tree can recursively embed into other trees. These are called *auxiliary trees.* The non-recursive trees are called *initial trees.* The word that selects a tree is called the *anchor* of that tree. In Figure 1.5 the set of trees anchored by each word in this sentence is shown. Parsing involves the attachment of nodes in each of these trees into the appropriate node in another tree. These attachments are categorized into two operations in the formalism. For example, the parse for the example sentence shown in Figure 1.7 is constructed using the following steps. In Figure 1.6, the tree **1[Ms.]** is an auxiliary tree that replaces the rootnode of tree **0[Haag]** with its root and foot node: this is termed as adjunction. This operation forms the constituent *Ms. Haag* in Figure 1.7. Replacing the *NParg* nodes in the frontier of the tree **7[plays]** for *plays* with the trees **0[Haag]** and **0[Elianti]** is termed as substitution. This completes the parse of the sentence resulting in the parse tree shown in Figure 1.7.

In this thesis, we will characterize both the operations of substitution and adjunction simply as one term: **attachment**.

Notice that as a consequence of this kind of lexicalized grammatical description there might be several different factors that affect the probability models used for statistical parsing as well as the complexity of parsing using such a formalism. Each word can select many different trees; for example, the word *plays* in Figure 1.5 might select several other trees not shown in that figure for each syntactic context in which it can occur. The verb *plays* can be used in a relative clause, a wh-extraction clause, among others. While grammatical notions of argument structure and syntax can be processed in abstract terms just as in other kinds of formalisms, the crucial difference in LTAG is that all of this information is compiled into a finite set of trees *before*

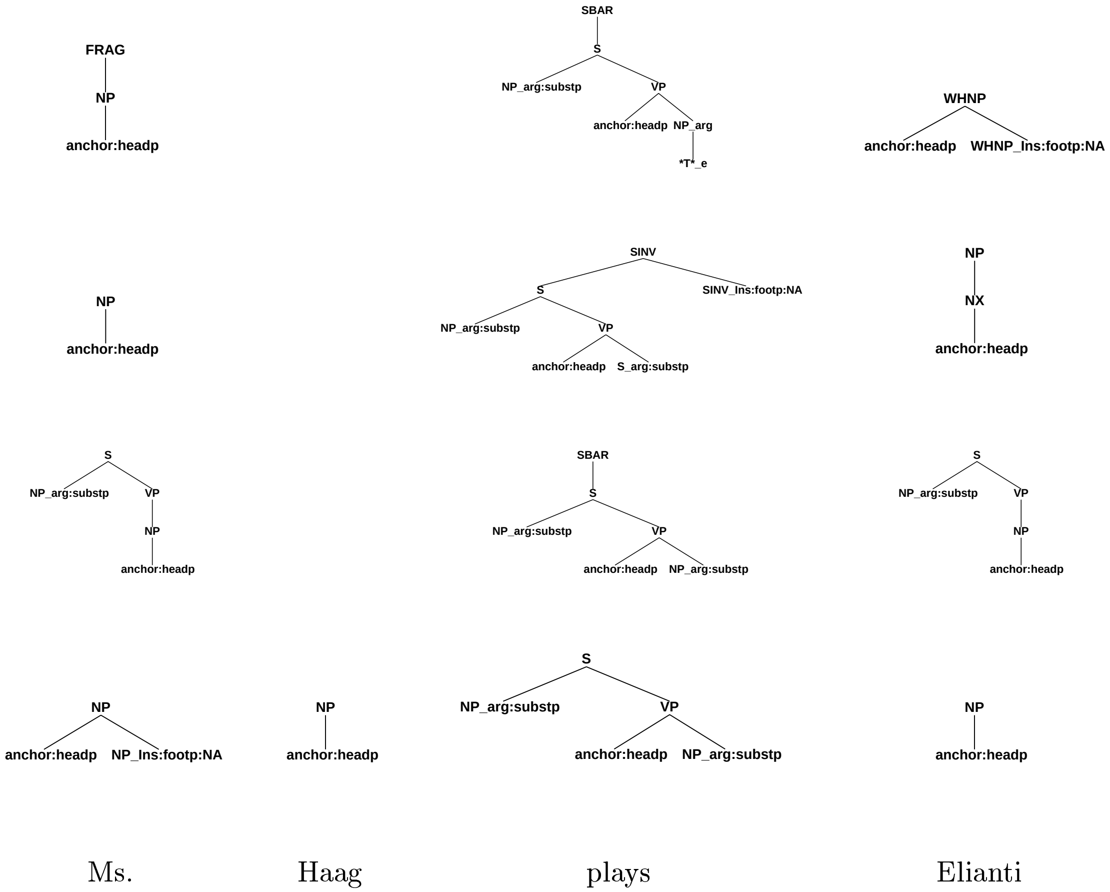


Figure 1.5: Trees selected by the sentence: *Ms. Haag plays Elianti*.

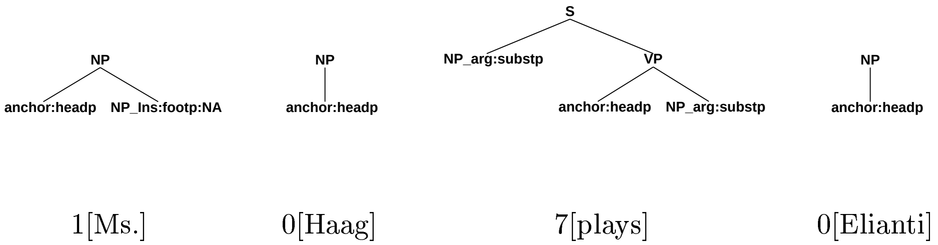


Figure 1.6: Trees used in the derivation of *Ms. Haag plays Elianti*.

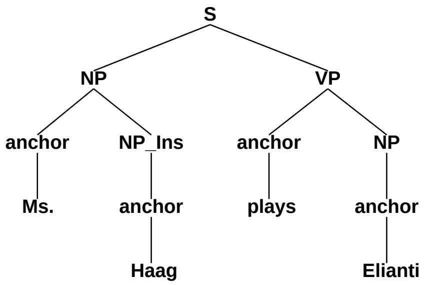


Figure 1.7: Parse tree produced by finding a consistent set of attachments that covers the sentence.

parsing. Each of these separate lexicalized trees is now considered by the parser. This compilation is repeated for other argument structures, e.g. the verb *plays* could also select trees which are intransitive thus increasing the set of lexicalized trees it can select. The set of trees selected by different lexical items is what we term in this thesis as *lexical syntactic ambiguity*.

The importance of this compilation into a set of lexicalized trees is that each predicate-argument structure across each syntactic context has its own lexicalized tree. Other parsers using different grammar formalisms use either articulated and complex probability models or alternatively they use feature structures to capture the same grammatical and predicate-argument information. In LTAG, this larger set of lexicalized trees directly corresponds to the fact that additions to the probability model that interact with recursion or recursive feature structures are not needed for linguistic description.

We argue in this thesis that the result of having compiled out abstract grammatical descriptions into a set of lexicalized trees allows us to simplify the probability models used for statistical parsing. We argue that it allows us to explore novel methods of dealing with learning a parser that are difficult to consider in other formalisms.

A stochastic LTAG derivation proceeds as follows [Schabes, 1992, Resnik, 1992]. An initial tree is selected with probability P_{INIT} and subsequent substitutions and adjunctions are performed each with probability P_{ATTACH}.

For each τ that can be valid start of a derivation:

$$\sum_{\tau} P_{\text{INIT}}(\tau) = 1$$

Each subsequent attachment occurs independently, with the condition that:

$$\sum_{\tau'} P_{\text{ATTACH}}(\tau, \eta \rightarrow \tau') = 1$$

where, τ' is substituting or adjoining into node η in tree τ. For adjunctions there is an additional factor which is required for the probability to be well-formed:

$$P_{\text{ATTACH}}(\tau, \eta \rightarrow \textsc{na})$$

which is the probability that there is no adjunction (NA) at node η in τ.

Each LTAG derivation $\mathcal{D}$ is built starting from some tree α. Let us consider the probability of a derivation $\mathcal{D}$ where each elementary tree has exactly one word w_i in its frontier from the input sentence $w_0 \ldots w_n$. Let m be the number of nodes for each elementary tree in the derivation where an attachment could have occurred.

$$\begin{aligned} & Pr(\mathcal{D}, w_0 \ldots w_n) = && (1.1) \\ & \quad P_{\text{INIT}}(\alpha, w_i) \times \prod_{0 \le k \le n, k \ne i} P_{\text{ATTACH}}(\tau, \eta, w \rightarrow \tau', w_k) \times \\ & \quad \prod_{m-k} P_{\text{ATTACH}}(\tau, \eta, w \rightarrow \textsc{na}) && (1.2) \end{aligned}$$

P_{INIT} and P_{ATTACH} can be written as the following conditional probabilities which can be estimated from the training data.

$$\begin{aligned} & P_{\text{INIT}}(\alpha, w_i) \\ & \quad = \Pr(\text{TOP} \rightarrow \alpha, w_i) \\ & \quad = \Pr(\alpha, w_i \mid \text{TOP}) \end{aligned}$$

$$P_{\mathrm{ATTACH}}(\tau, \eta, w \rightarrow \tau', w_k) = \Pr(\tau', w_k \mid \tau, \eta, w)$$

This derivation $\mathcal{D}$ can be drawn graphically as a tree where each node in this derivation tree is an elementary tree in the original LTAG. For example, combining the lowermost trees for each word in Figure 1.5 gives us the derivation tree in Figure 1.8. By combining each of the elementary trees we can also draw the usual constituent structure shown in Figure 1.7. This structure can be further simplified by removing certain nodes that are in the constituent structure as a result of the adjunction operation. This final parse is shown in Figure 1.9.

The probability of a sentence S computed using this model is the sum of all the possible derivations of the sentence.

$$P(S) = \sum_{D} Pr(D, S)$$

A generative model can be defined instead of a conditional probability to obtain the best derivation $\mathcal{D}_{\mathrm{BEST}}$ given a sentence S. The value for (1.3) is computed using the Equation 1.1.

$$\begin{aligned}
\mathcal{D}_{\mathrm{BEST}} &= \arg\max_{\mathcal{D}} \Pr(\mathcal{D} \mid S) \\
&= \arg\max_{\mathcal{D}} \frac{\Pr(\mathcal{D}, S)}{\Pr(S)} \\
&= \arg\max_{\mathcal{D}} \Pr(\mathcal{D}, S) \qquad (1.3)
\end{aligned}$$

1.6 Publication History

Various parts of this dissertation proposal have been published in conferences. I would like to list these publications here as well as acknowledge my co-authors in this work.

- The result in Chapter 2 was published as:

  Consistency of Probabilistic Tree Adjoining Grammars. In Proceedings of COLING-ACL 1998, Montreal.

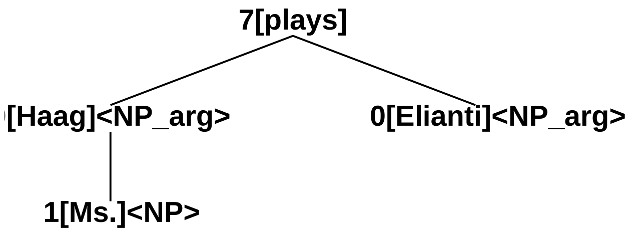


Figure 1.8: A derivation indicating all the attachments between trees that have occurred during the parse of the sentence.

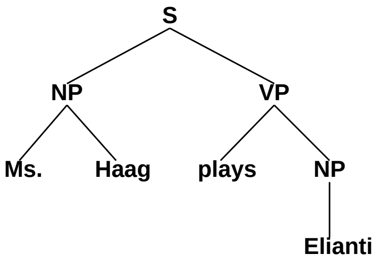


Figure 1.9: Parse tree after all nodes that were inserted for parsing are removed.

- The part of Chapter 5 which deals with the parsing of initial substring was published as:

  Probabilities from Probabilistic Tree Adjoining Grammars. Mark-Jan Nederhof, Anoop Sarkar and Giorgio Satta. In Proceedings of COLING-ACL 1998, Montreal.

- In Chapter C the resuls on subcategorization learning was published as:

  Automatic Extraction of Subcategorization Frames for Czech. Anoop Sarkar and Daniel Zeman. In Proceedings of COLING-2000. Saarbruecken, Germany, August 2000.

Chapter 2

Conditions on Consistency of Probabilistic TAGs

Much of the power of probabilistic methods in modelling language comes from their ability to compare several derivations for the same string in the language. This cross-derivational power arises naturally from comparison of various derivational paths, each of which is a product of the probabilities associated with each step in each derivation. A common approach used to assign structure to language is to use a probabilistic grammar where each elementary rule or production is associated with a probability. Using such a grammar, a probability for each string in the language is computed. Assuming that the probability of each derivation of a sentence is well-defined, the probability of each string in the language is simply the sum of the probabilities of all derivations of the string. In general, for a probabilistic grammar G the language of G is denoted by $L(G)$. Then if a string v is in the language $L(G)$ the probabilistic grammar assigns v some non-zero probability.

There are several cross-derivational properties that can be studied for a given probabilistic grammar formalism. An important starting point for such studies is the notion of *consistency*. The probability model defined by a probabilistic grammar is said to be *consistent* if the probabilities assigned to all the strings in the language sum to 1. That is, if Pr defined by a probabilistic grammar, assigns a probability to each string $v \in \Sigma^*$, where $\Pr(v) = 0$ if $v \notin L(G)$, then

$$\sum_{v \in L(G)} \Pr(v) = 1 \tag{2.1}$$

From the literature on probabilistic context-free grammars (CFGs) we know precisely the conditions which ensure that (2.1) is true for a given CFG. This chapter derives the conditions under which a given probabilistic TAG can be shown to be consistent.

TAGs are important in the modelling of natural language since they can be easily lexicalized; moreover the trees associated with words can be used to encode argument and adjunct relations in various syntactic environments. [Joshi, 1988] and [Joshi and Schabes, 1992] are good introductions to the formalism and its linguistic relevance. TAGs have been shown to have relations with both phrase-structure grammars and dependency grammars [Rambow and Joshi, 1995] and can handle (non-projective) long distance dependencies.

Consistency of probabilistic TAGs has practical significance for the following reasons:

- The conditions derived here can be used to ensure that probability models that use TAGs can be checked for *deficiency*.

- Existing EM based estimation algorithms for probabilistic TAGs assume that the property of consistency holds [Schabes, 1992]. EM based algorithms begin with an initial (usually random) value for each parameter. If the initial assignment causes the grammar to be inconsistent, then iterative re-estimation might converge to an inconsistent grammar[1].

- Techniques used in this chapter can be used to determine consistency for other probability models based on TAGs [Carroll and Weir, 1997].

2.1 Notation

In this section we establish some notational conventions and definitions that we use in this chapter. Those familiar with the TAG formalism only need to give a cursory glance

[1] Note that for CFGs it has been shown in [Chaudhari et al., 1983, Sánchez and Benedí, 1997] that inside-outside reestimation can be used to avoid inconsistency. We will show later in the chapter that the method used to show consistency in this chapter precludes a straightforward extension of that result for TAGs.

through this section.

A probabilistic TAG is represented by $(N, \Sigma, \mathcal{I}, \mathcal{A}, S, \phi)$ where N, Σ are, respectively, non-terminal and terminal symbols. $\mathcal{I} \cup \mathcal{A}$ is a set of trees termed as *elementary trees*. We take V to be the set of all nodes in all the elementary trees. For each leaf $A \in V$, $label(A)$ is an element from $\Sigma \cup \{\epsilon\}$, and for each other node A, $label(A)$ is an element from N. S is an element from N which is a distinguished start symbol. The root node A of every initial tree which can start a derivation must have $label(A) = S$.

$\mathcal{I}$ are termed *initial trees* and $\mathcal{A}$ are *auxiliary trees* which can rewrite a tree node $A \in V$. This rewrite step is called **adjunction**. ϕ is a function which assigns each adjunction with a probability and denotes the set of parameters in the model. In practice, TAGs also allow a leaf nodes A such that $label(A)$ is an element from N. Such nodes A are rewritten with initial trees from $\mathcal{I}$ using the rewrite step called **substitution**. Except in one special case, we will not need to treat substitution as being distinct from adjunction.

For $t \in \mathcal{I} \cup \mathcal{A}$, $\mathcal{A}(t)$ are the nodes in tree t that can be modified by adjunction. For $label(A) \in N$ we denote $Adj(label(A))$ as the set of trees that can adjoin at node $A \in V$. The adjunction of t into $N \in V$ is denoted by $N \mapsto t$. No adjunction at $N \in V$ is denoted by $N \mapsto nil$. We assume the following properties hold for every probabilistic TAG G that we consider:

1. G is *lexicalized*. There is at least one leaf node a that lexicalizes each elementary tree, i.e. $a \in \Sigma$.

2. G is *proper*. For each $N \in V$,

$$\phi(N \mapsto nil) + \sum_t \phi(N \mapsto t) = 1$$

3. Adjunction is prohibited on the foot node of every auxiliary tree. This condition is imposed to avoid unnecessary ambiguity and can be easily relaxed.

4. There is a distinguished non-lexicalized initial tree τ such that each initial tree rooted by a node A with $label(A) = S$ substitutes into τ to complete the derivation. This ensures that probabilities assigned to the input string at the start of the derivation are well-formed.

We use symbols $S, A, B, \ldots$ to range over V, symbols $a, b, c, \ldots$ to range over Σ. We use $t_1, t_2, \ldots$ to range over $I \cup A$ and ϵ to denote the empty string. We use X_i to range over all i nodes in the grammar.

2.2 Applying probability measures to Tree Adjoining Languages

To gain some intuition about probability assignments to languages, let us take for example, a language well known to be a tree adjoining language:

$$L(G) = \{a^n b^n c^n d^n | n \geq 1\}$$

It seems that we should be able to use a function ψ to assign any probability distribution to the strings in $L(G)$ and then expect that we can assign appropriate probabilites to the adjunctions in G such that the language generated by G has the same distribution as that given by ψ. However a function ψ that grows smaller by repeated multiplication as the inverse of an exponential function cannot be matched by any TAG because of the *constant growth* property of TAGs (see [Vijay-Shanker, 1987], p. 104). An example of such a function ψ is a simple Poisson distribution (2.2), which in fact was also used as the counterexample in [Booth and Thompson, 1973] for CFGs, since CFGs also have the constant growth property.

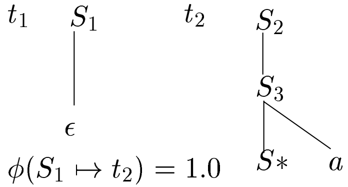
$$\psi(a^n b^n c^n d^n) = \frac{1}{e \cdot n!} \tag{2.2}$$

This shows that probabilistic TAGs, like CFGs, are constrained in the probabilistic languages that they can recognize or learn. As shown above, a probabilistic language can fail to have a generating probabilistic TAG.

The reverse is also true: some probabilistic TAGs, like some CFGs, fail to have a corresponding probabilistic language, i.e. they are not consistent. There are two reasons why a probabilistic TAG could be inconsistent: "dirty" grammars, and destructive or incorrect probability assignments.

"Dirty" grammars. Usually, when applied to language, TAGs are lexicalized and so probabilities assigned to trees are used only when the words anchoring the trees are used in a derivation. However, if the TAG allows non-lexicalized trees, or more precisely, auxiliary trees with no yield, then looping adjunctions which never generate a string are possible. However, this can be detected and corrected by a simple search over the grammar. Even in lexicalized grammars, there could be some auxiliary trees that are assigned some probability mass but which can never adjoin into another tree. Such auxiliary trees are termed *unreachable* and techniques similar to the ones used in detecting unreachable productions in CFGs can be used here to detect and eliminate such trees.

Destructive probability assignments. This problem is a more serious one, and is the main subject of this chapter. Consider the probabilistic TAG shown in (2.3)[2].



$$\phi(S_1 \mapsto t_2) = 1.0$$

$$\begin{array}{l} \phi(S_2 \mapsto t_2) = 0.99 \\ \phi(S_2 \mapsto nil) = 0.01 \\ \phi(S_3 \mapsto t_2) = 0.98 \\ \phi(S_3 \mapsto nil) = 0.02 \end{array} \tag{2.3}$$

Consider a derivation in this TAG as a generative process. It proceeds as follows: node S_1 in t_1 is rewritten as t_2 with probability 1.0. Node S_2 in t_2 is 99 times more likely than not to be rewritten as t_2 itself, and similarly node S_3 is 49 times more likely than not to be rewritten as t_2. This however, creates two more instances of S_2 and S_3 with same probabilities. This continues, creating multiple instances of t_2 at each level of the derivation process with each instance of t_2 creating two more instances of itself. The grammar itself is not malicious; the probability assignments are to blame. It is important to note that inconsistency is a problem even though for any given string there are only a finite number of derivations, all halting. Consider the probability mass function (*pmf*) over the set of all derivations for this grammar. An inconsistent grammar would have a *pmf* which assigns a large portion of probability mass to derivations that are non-terminating. This means there is a finite probability the generative process can enter a generation sequence which has a finite probability of non-termination.

[2] The subscripts are used as a simple notation to uniquely refer to the nodes in each elementary tree. They are not part of the node label for purposes of adjunction.

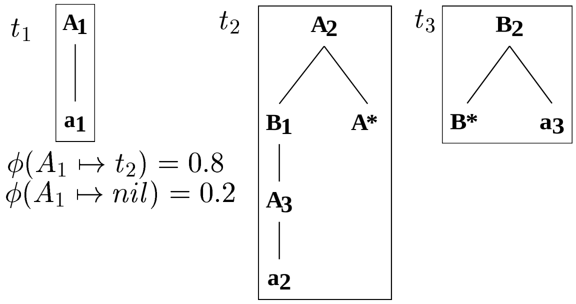
2.3 Conditions for Consistency

A probabilistic TAG G is *consistent* if and only if:

$$\sum_{v \in L(G)} \Pr(v) = 1 \tag{2.4}$$

where $\Pr(v)$ is the probability assigned to a string in the language. If a grammar G does not satisfy this condition, G is said to be inconsistent.

To explain the conditions under which a probabilistic TAG is consistent we will use the TAG in (2.5) as an example.



$$\begin{aligned} &\phi(A_1 \mapsto t_2) = 0.8 \\ &\phi(A_1 \mapsto nil) = 0.2 \end{aligned}$$

$$\begin{aligned} &\phi(A_2 \mapsto t_2) = 0.2 && \phi(B_2 \mapsto t_3) = 0.1 \\ &\phi(A_2 \mapsto nil) = 0.8 && \phi(B_2 \mapsto nil) = 0.9 \\ &\phi(B_1 \mapsto t_3) = 0.2 \\ &\phi(B_1 \mapsto nil) = 0.8 \\ &\phi(A_3 \mapsto t_2) = 0.4 \\ &\phi(A_3 \mapsto nil) = 0.6 \end{aligned} \tag{2.5}$$

From this grammar, we compute a square matrix $\mathcal{M}$ which of size $|V|$, where V is the set of nodes in the grammar that can be rewritten by adjunction. Each $\mathcal{M}_{ij}$ contains the expected value of obtaining node X_j when node X_i is rewritten by adjunction at each level of a TAG derivation. We call $\mathcal{M}$ the stochastic *expectation matrix* associated with a probabilistic TAG.

To get $\mathcal{M}$ for a grammar we first write a matrix $\mathbf{P}$ which has $|V|$ rows and $|I \cup A|$ columns. An element $\mathbf{P}_{ij}$ corresponds to the probability of adjoining tree t_j at node X_i, i.e. $\phi(X_i \mapsto t_j)$[3].

[3] Note that $\mathbf{P}$ is not a row stochastic matrix. This is an important difference in the construction of $\mathcal{M}$ for TAGs when compared to CFGs. We will return to this point in §2.4.

$$\mathbf{P} = \begin{array}{c} \\ A_1 \\ A_2 \\ B_1 \\ A_3 \\ B_2 \end{array} \begin{array}{c} \begin{array}{ccc} t_1 & t_2 & t_3 \end{array} \\ \left[\begin{array}{ccc} 0 & 0.8 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.2 \\ 0 & 0.4 & 0 \\ 0 & 0 & 0.1 \end{array}\right] \end{array}$$

We then write a matrix $\mathbf{N}$ which has $|I \cup A|$ rows and $|V|$ columns. An element $\mathbf{N}_{ij}$ is 1.0 if node X_j is a node in tree t_i.

$$\mathbf{N} = \begin{array}{c|ccccc} & A_1 & A_2 & B_1 & A_3 & B_2 \\ \hline t_1 & 1.0 & 0 & 0 & 0 & 0 \\ t_2 & 0 & 1.0 & 1.0 & 1.0 & 0 \\ t_3 & 0 & 0 & 0 & 0 & 1.0 \end{array}$$

Then the stochastic expectation matrix $\mathcal{M}$ is simply the product of these two matrices.

$$\mathcal{M} = \mathbf{P} \cdot \mathbf{N} = \begin{array}{c|ccccc} & A_1 & A_2 & B_1 & A_3 & B_2 \\ \hline A_1 & 0 & 0.8 & 0.8 & 0.8 & 0 \\ A_2 & 0 & 0.2 & 0.2 & 0.2 & 0 \\ B_1 & 0 & 0 & 0 & 0 & 0.2 \\ A_3 & 0 & 0.4 & 0.4 & 0.4 & 0 \\ B_2 & 0 & 0 & 0 & 0 & 0.1 \end{array}$$

By inspecting the values of $\mathcal{M}$ in terms of the grammar probabilities indicates that $\mathcal{M}_{ij}$ contains the values we wanted, i.e. expectation of obtaining node A_j when node A_i is rewritten by adjunction at each level of the TAG derivation process.

By construction we have ensured that the following theorem from [Booth and Thompson, 1973] applies to probabilistic TAGs. A formal justification for this claim is given in the next section by showing a reduction of the TAG derivation process to a multitype Galton-Watson branching process [Harris, 1963].

Theorem 2.3.1 *A probabilistic grammar is consistent if the* spectral radius $\rho(\mathcal{M}) < 1$, *where $\mathcal{M}$ is the stochastic expectation matrix computed from the grammar. [Booth and Thompson, 1973, Soule, 1974]*

This theorem provides a way to determine whether a grammar is consistent. All we need to do is compute the spectral radius of the square matrix $\mathcal{M}$ which is equal to the modulus of the largest eigenvalue of $\mathcal{M}$. If this value is less than one then the grammar is consistent[4]. Computing consistency can bypass the computation of the eigenvalues for $\mathcal{M}$ by using the following theorem by Geršgorin (see [Horn and Johnson, 1985, Wetherell, 1980]).

Theorem 2.3.2 *For any square matrix $\mathcal{M}$, $\rho(\mathcal{M}) < 1$ if and only if there is an $n \geq 1$ such that the sum of the absolute values of the elements of each row of $\mathcal{M}^n$ is less than one. Moreover, any $n' > n$ also has this property. (Geršgorin, see [Horn and Johnson, 1985, Wetherell, 1980])*

This makes for a very simple algorithm to check consistency of a grammar. We sum the values of the elements of each row of the stochastic expectation matrix $\mathcal{M}$ computed from the grammar. If *any* of the row sums are greater than one then we compute $\mathcal{M}^2$, repeat the test and compute $\mathcal{M}^{2^2}$ if the test fails, and so on until the test succeeds[5]. The algorithm does not halt if $\rho(\mathcal{M}) \geq 1$. In practice, such an algorithm works better in the average case since computation of eigenvalues is more expensive for very large matrices. An upper bound can be set on the number of iterations in this algorithm. Once the bound is passed, the exact eigenvalues can be computed.

For the grammar in (2.5) we computed the following stochastic expectation matrix:

$$\mathcal{M} = \begin{bmatrix} 0 & 0.8 & 0.8 & 0.8 & 0 \\ 0 & 0.2 & 0.2 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 0.2 \\ 0 & 0.4 & 0.4 & 0.4 & 0 \\ 0 & 0 & 0 & 0 & 0.1 \end{bmatrix}$$

[4] The grammar may be consistent when the spectral radius is exactly one, but this case involves many special considerations and is not considered in this chapter. In practice, these complicated tests are probably not worth the effort. See [Harris, 1963] for details on how this special case can be solved.

[5] We compute $\mathcal{M}^{2^2}$ and subsequently only successive powers of 2 because Theorem 2.3.2 holds for any $n' > n$. This permits us to use a single matrix at each step in the algorithm.

The first row sum is 2.4. Since the sum of each row must be less than one, we compute the power matrix $\mathcal{M}^2$. However, the sum of one of the rows is still greater than 1. Continuing we compute $\mathcal{M}^{2^2}$.

$$\mathcal{M}^{2^2} = \begin{bmatrix} 0 & 0.1728 & 0.1728 & 0.1728 & 0.0688 \\ 0 & 0.0432 & 0.0432 & 0.0432 & 0.0172 \\ 0 & 0 & 0 & 0 & 0.0002 \\ 0 & 0.0864 & 0.0864 & 0.0864 & 0.0344 \\ 0 & 0 & 0 & 0 & 0.0001 \end{bmatrix}$$

This time all the row sums are less than one, hence $\rho(\mathcal{M}) < 1$. So we can say that the grammar defined in (2.5) is consistent. We can confirm this by computing the eigenvalues for $\mathcal{M}$ which are $0, 0, 0.6, 0$ and 0.1, all less than 1.

Now consider the grammar (2.3) we had considered in Section 2.2. The value of $\mathcal{M}$ for that grammar is computed to be:

$$\mathcal{M}_{(2.3)} = \begin{array}{c} \\ S_1 \\ S_2 \\ S_3 \end{array} \begin{array}{c} \begin{array}{ccc} S_1 & S_2 & S_3 \end{array} \\ \begin{bmatrix} 0 & 1.0 & 1.0 \\ 0 & 0.99 & 0.99 \\ 0 & 0.98 & 0.98 \end{bmatrix} \end{array}$$

The eigenvalues for the expectation matrix $\mathcal{M}$ computed for the grammar (2.3) are 0, 1.97 and 0. The largest eigenvalue is greater than 1 and this confirms (2.3) to be an inconsistent grammar.

2.4 TAG Derivations and Branching Processes

To show that Theorem 2.3.1 in Section 2.3 holds for any probabilistic TAG, it is sufficient to show that the derivation process in TAGs is a Galton-Watson branching process.

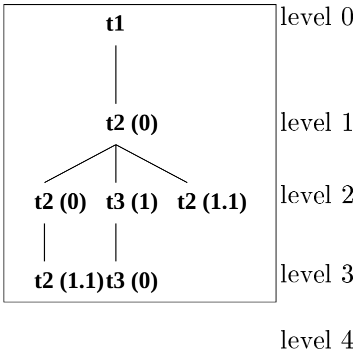
A Galton-Watson branching process [Harris, 1963] is simply a model of processes that have objects that can produce additional objects of the same kind, i.e. recursive processes, with certain properties. There is an initial set of objects in the 0-th generation which produces with some probability a first generation which in turn with some probability generates a second, and so on. We will denote by vectors $Z_0, Z_1, Z_2, \ldots$ the 0-th, first, second, ... generations. There are two assumptions made about $Z_0, Z_1, Z_2, \ldots$:

1. The size of the n-th generation does not influence the probability with which any of the objects in the $(n+1)$-th generation is produced. In other words, $Z_0, Z_1, Z_2, \ldots$ form a Markov chain.

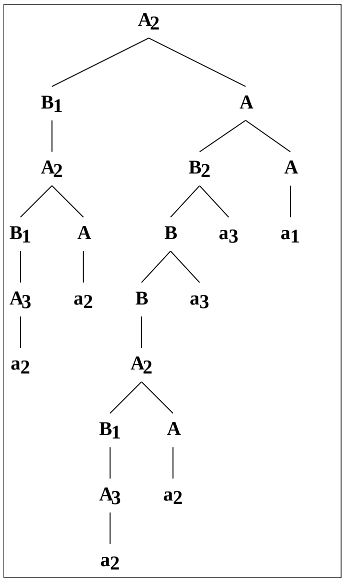
2. The number of objects born to a parent object does not depend on how many other objects are present at the same level.

We can associate a generating function for each level Z_i. The value for the vector Z_n is the value assigned by the n-th iterate of this generating function. The expectation matrix $\mathcal{M}$ is defined using this generating function.

The theorem attributed to Galton and Watson specifies the conditions for the probability of extinction of a family starting from its 0-th generation, assuming the branching process represents a family tree (i.e, respecting the conditions outlined above). The theorem states that $\rho(\mathcal{M}) \leq 1$ when the probability of extinction is 1.0.



(2.6)



(2.7)

The assumptions made about the generating process intuitively holds for probabilistic TAGs. (2.6), for example, depicts a derivation of the string $a_2a_2a_2a_2a_3a_3a_1$ by a sequence of adjunctions in the grammar given in (2.5)[6]. The parse tree derived from such a sequence is shown in Fig. 2.7. In the derivation tree (2.6), nodes in the trees at each level i are rewritten by adjunction to produce a level $i+1$. There is a final level 4 in (2.6) since we also consider the probability that a node is not rewritten further, i.e. $\Pr(A \mapsto nil)$ for each node A.

We give a precise statement of a TAG derivation process by defining a generating function for the levels in a derivation tree. Each level i in the TAG derivation tree then corresponds to Z_i in the Markov chain of branching processes. This is sufficient to justify the use of Theorem 2.3.1 in Section 2.3. The conditions on the probability of extinction then relates to the probability that TAG derivations for a probabilistic TAG will not recurse infinitely. Hence the probability of extinction is the same as the probability that a probabilistic TAG is consistent.

For each $X_j \in V$, where V is the set of nodes in the grammar where adjunction can occur, we define the k-argument *adjunction generating function* over variables $s_1, \ldots, s_k$

[6] The numbers in parentheses next to the tree names are node addresses where each tree has adjoined into its parent. Recall the definition of node addresses in Section 2.1.

corresponding to the k nodes in V.

$$g_j(s_1, \ldots, s_k) = \sum_{t \in Adj(X_j) \cup \{nil\}} \phi(X_j \mapsto t) \cdot s_1^{r_1(t)} \cdots s_k^{r_k(t)}$$

where, $r_j(t) = 1$ iff node X_j is in tree t, $r_j(t) = 0$ otherwise.

For example, for the grammar in (2.5) we get the following adjunction generating functions taking the variable s_1, s_2, s_3, s_4, s_5 to represent the nodes A_1, A_2, B_1, A_3, B_2 respectively.

$$g_1(s_1, \ldots, s_5) = \phi(A_1 \mapsto t_2) \cdot s_2 \cdot s_3 \cdot s_4 + \phi(A_1 \mapsto nil)$$

$$g_2(s_1, \ldots, s_5) = \phi(A_2 \mapsto t_2) \cdot s_2 \cdot s_3 \cdot s_4 + \phi(A_2 \mapsto nil)$$

$$g_3(s_1, \ldots, s_5) = \phi(B_1 \mapsto t_3) \cdot s_5 + \phi(B_1 \mapsto nil)$$

$$g_4(s_1, \ldots, s_5) = \phi(A_3 \mapsto t_2) \cdot s_2 \cdot s_3 \cdot s_4 + \phi(A_3 \mapsto nil)$$

$$g_5(s_1, \ldots, s_5) = \phi(B_2 \mapsto t_3) \cdot s_5 + \phi(B_2 \mapsto nil)$$

The n-th level generating function $G_n(s_1, \ldots, s_k)$ is defined recursively as follows.

$$\begin{aligned} G_0(s_1, \ldots, s_k) &= s_1 \\ G_1(s_1, \ldots, s_k) &= g_1(s_1, \ldots, s_k) \\ G_n(s_1, \ldots, s_k) &= G_{n-1}[g_1(s_1, \ldots, s_k), \ldots, \\ &\quad g_k(s_1, \ldots, s_k)] \end{aligned}$$

For the grammar in (2.5) we get the following level generating functions.

$$G_0(s_1, \ldots, s_5) = s_1$$

$$G_1(s_1, \ldots, s_5) = g_1(s_1, \ldots, s_5)$$

$$
\begin{aligned}
&= \phi(A_1 \mapsto t_2) \cdot s_2 \cdot s_3 \cdot s_4 + \phi(A_1 \mapsto nil) \\
&= 0.8 \cdot s_2 \cdot s_3 \cdot s_4 + 0.2 \\
G_2&(s_1, \ldots, s_5) = \\
&\quad \phi(A_2 \mapsto t_2)[g_2(s_1, \ldots, s_5)][g_3(s_1, \ldots, s_5)] \\
&\qquad [g_4(s_1, \ldots, s_5)] + \phi(A_2 \mapsto nil) \\
&= 0.08 s_2^2 s_3^2 s_4^2 s_5 + 0.03 s_2^2 s_3^2 s_4^2 + 0.04 s_2 s_3 s_4 s_5 + \\
&\qquad 0.18 s_2 s_3 s_4 + 0.04 s_5 + 0.196 \\
&\ldots
\end{aligned}
$$

Examining this example, we can express $G_i(s_1, \ldots, s_k)$ as a sum $D_i(s_1, \ldots, s_k) + C_i$, where C_i is a constant and $D_i(\cdot)$ is a polynomial with no constant terms. A probabilistic TAG will be consistent if these recursive equations terminate, i.e. iff

$$lim_{i \to \infty} D_i(s_1, \ldots, s_k) \to 0$$

We can rewrite the level generation functions in terms of the stochastic expectation matrix $\mathcal{M}$, where each element $m_{i,j}$ of $\mathcal{M}$ is computed as follows (cf. [Booth and Thompson, 1973]).

$$m_{i,j} = \left. \frac{\partial g_i(s_1, \ldots, s_k)}{\partial s_j} \right|_{s_1, \ldots, s_k = 1} \tag{2.8}$$

The limit condition above translates to the condition that the spectral radius of $\mathcal{M}$ must be less than 1 for the grammar to be consistent.

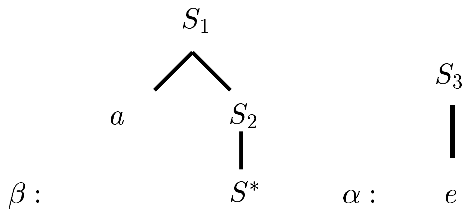
This shows that Theorem 2.3.1 used in Section 2.3 to give an algorithm to detect inconsistency in a probabilistic holds for any given TAG, hence demonstrating the correctness of the algorithm.

Note that the formulation of the adjunction generating function means that the values for $\phi(X \mapsto nil)$ for all $X \in V$ do not appear in the expectation matrix. This is a crucial difference between the test for consistency in TAGs as compared to CFGs. For CFGs, the expectation matrix for a grammar G can be interpreted as the contribution of each non-terminal to the derivations for a sample set of strings drawn from $L(G)$. Using this it was shown in [Chaudhari et al., 1983] and [Sánchez and Benedí, 1997] that a single step of the inside-outside algorithm implies consistency for a probabilistic CFG. In the next section we will give an alternative method that exploits the context-free nature of TAG derivations. This will allow us to leverage earlier results shown for probabilistic CFGs.

2.5 Alternative Method: Reducing TAG derivations to CFGs

In this section we will give an alternative method that exploits the context-free nature of TAG derivations. This method was suggested to us by Steve Abney as an alternative to the approach given in the earlier sections.

We explain the alternative method using an example. Consider a PTAG $G = \langle \{\beta, \alpha\}, \phi \rangle$ with trees defined in (2.9) and parameter values given in (2.10).



(2.9)

$$\begin{aligned}
\phi(S_1 \rightarrow \beta) &= 0.99 \\
\phi(S_1 \rightarrow \epsilon) &= 0.01 \\
\phi(S_2 \rightarrow \beta) &= 0.98 \\
\phi(S_2 \rightarrow \epsilon) &= 0.02 \\
\phi(S_3 \rightarrow \beta) &= 1.0 \\
\phi(S_3 \rightarrow \epsilon) &= 0.0
\end{aligned} \tag{2.10}$$

G is an inconsistent PTAG (it assigns probability mass to derivations that do not terminate). *Can we detect this inconsistency by using the Booth and Thompson result on some PCFG G' constructed by examining the PTAG G ?*

In (2.11) we show a set of productions (rules) P constructed from the PTAG G designed to show the derivations possible in G.

$$\alpha \rightarrow S_3 \tag{2.11}$$

$$\begin{array}{rcl} S_3 & \rightarrow & \beta \mid \epsilon \\ \beta & \rightarrow & S_1\, S_2 \\ S_1 & \rightarrow & \beta \mid \epsilon \\ S_2 & \rightarrow & \beta \mid \epsilon \end{array}$$

To use the Booth and Thompson result we need to satisfy two conditions. The first condition is that the PCFG is *proper*. That is the parameters of the PCFG satisfy Eqn. 2.12.

$$\sum_{(X \rightarrow Y) \in P} \mathcal{P}(X \rightarrow Y \mid X) = 1 \tag{2.12}$$

The rule probabilities for the CFG in (2.11) derived from the PTAG parameters in (2.10) are shown in (2.13).

$$\begin{array}{rcl} \mathcal{P}(\alpha \rightarrow S_3 \mid \alpha) & = & \sum_X \mathcal{P}(S_3 \rightarrow X \mid S_3) = 1.0 \\ \mathcal{P}(S_3 \rightarrow \beta \mid S_3) & = & 1.0 \\ \mathcal{P}(S_3 \rightarrow \epsilon \mid S_3) & = & 0.0 \\ \mathcal{P}(\beta \rightarrow S_1\, S_2 \mid \beta) & = & \sum_X \mathcal{P}(S_1 \rightarrow X \mid S_1) \times \sum_Y \mathcal{P}(S_2 \rightarrow Y \mid S_2) = 1.0 \\ \mathcal{P}(S_1 \rightarrow \beta \mid S_1) & = & 0.99 \\ \mathcal{P}(S_1 \rightarrow \epsilon \mid S_1) & = & 0.01 \\ \mathcal{P}(S_2 \rightarrow \beta \mid S_2) & = & 0.98 \\ \mathcal{P}(S_2 \rightarrow \epsilon \mid S_2) & = & 0.02 \end{array} \tag{2.13}$$

Based on these probabilities we can compute the expectation matrix $\mathcal{M}$ using the method defined in Booth and Thompson.

$$\mathcal{M} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0.9900 \\ 0 & 0 & 0 & 0 & 0.9800 \\ 0 & 0 & 0 & 0 & 1.0000 \\ 0 & 0 & 1.0000 & 0 & 0 \\ 1.0000 & 1.0000 & 0 & 0 & 0 \end{bmatrix}$$

The eigenvalues of the expectation matrix come out to be $0, 1.4036, -1.4036$. Since the largest eigenvalue is greater than 1, we correctly predict that the original PTAG G is consistent. We get identical eigenvalues for this grammar using the original method presented earlier in this chapter. It remains to be seen whether the two methods are identical in all respects.

2.6 Conclusion

We have shown here that the conditions under which a given probabilistic TAG can be shown to be consistent. We gave a simple algorithm for checking consistency and gave the formal justification for its correctness. The result is practically significant for its applications in checking for *deficiency* in probabilistic TAGs. By leveraging the results shown for consistency in probabilistic CFGs and because of the context free nature of TAG derivations we can show the correctness of parameter estimation algorithms for probabilistic TAGs. And finally, the mathematical framework can be used to discover various statistical properties in a probabilistic TAG and the language it generates.

We also use the results shown in this chapter to compute the off-line equations that are required in Chapter 5 for prefix probability computation from probabilistic TAGs.

Chapter 3

A Head-Corner Parsing Algorithm for Probabilistic TAGs

In this chapter we define a parsing algorithm for probabilistic TAGs that we will use within the statistical parser we define in Chapter 4. The parser is a chart-based head-corner parser for general TAGs.

3.1 Parsing Algorithm

The parser introduced in this chapter implements a chart-based head-corner algorithm. The use of head-driven prediction to enchance efficiency was first suggested by [Kay, 1989] for CF parsing (see [Sikkel, 1997] for a more detailed survey). [Lavelli and Satta, 1991] provided the first head-driven algorithm for LTAGs which was a chart-based algorithm but it lacked any top-down prediction. [van Noord, 1994] describes a Prolog implementation of a head-corner parser for LTAGs which includes top-down prediction. Significantly, [van Noord, 1994] uses a different closure relation from [Lavelli and Satta, 1991]. The head-corner traversal for auxiliary trees starts from the footnode rather than from the anchor.

The parsing algorithm we use is a chart-based variant of the [van Noord, 1994] algorithm. We use the same head-corner closure relation as proposed there. Our parser differs from the algorithm in [van Noord, 1994] in some important respects: our implementation is chart-based and explicitly tracks *goal* and *item* states and does not perform any implicit backtracking or selective memoization, we do not need any additional variables to keep track of which words are already 'reserved' by an auxiliary tree (which [van Noord, 1994] needs to guarantee termination), and we have an explicit *completion* step. In addition, we introduce some optimizations as part of the parser definition. In this chapter we have chosen to present the algorithm in pseudo-code that is close to the actual implementation. The reason for this is that although a more compact mathematical description could be given, such a description does not emphasize several aspects of dynamic programming that are crucially needed to make parsing using this algorithm feasible, even if you ignore issues of efficient implementation.

3.2 Head-Corner Traversal

We take the concept of head-corner traversal for TAGs as defined in [van Noord, 1994]. We illustrate the head-corner closure relation using an example in this section and define it more formally later in this chapter.

Each node in an elementary tree in a TAG is associated with a distinguished node in the same tree called the *headcorner*. Parsing is initiated by making top-down predictions on certain nodes and proceeds by moving bottom-up from the headcorner associated with the goal node. This is done recursively producing new goal nodes for siblings and for adjunction prediction. For example, in Figure 3.1, parsing begins with the prediction that node *S/na* of tree T_1 will span the entire input sentence. The headcorner for node *S/na* is the terminal symbol *took*. The parser then proceeds from that node in the elementary tree T_1 moving up the tree to reach the goal node which is the root node of that tree. Each sibling node is generated as a new goal node before proceeding upwards with the parent of the node. In the figure, the dotted lines are traversed first, before traversing up to the parent node. In the example figure, after visiting the node *V/na* the node *N/top* is introduced as a new goal node which leads to the headcorner node *walk*. Any node where adjunction can occur is represented as two nodes for the parser: *Node/top* and *Node/bottom*. The adjunction is recognized between the bottom and top portions of

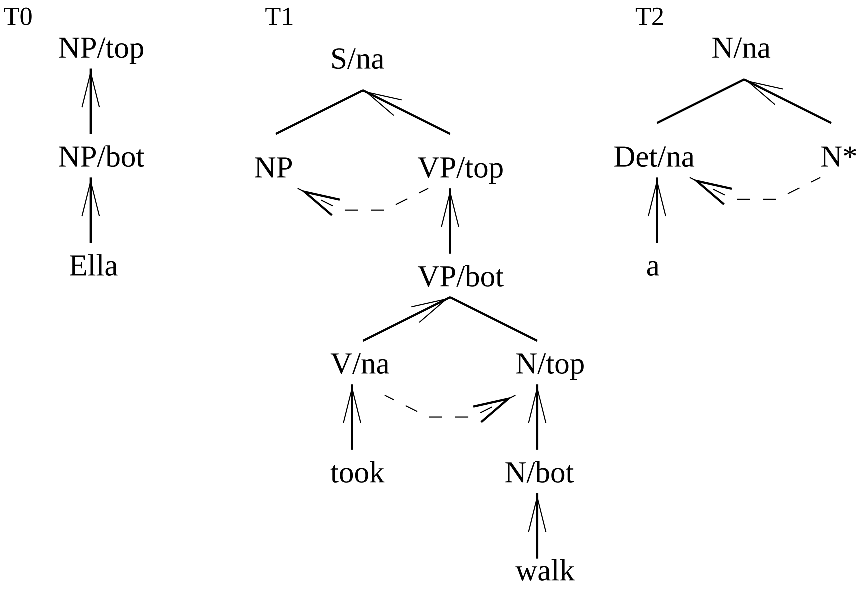


Figure 3.1: An example of head-corner traversal for parsing the sentence *Ella took a walk.*

the node. In cases where adjunction is prohibited the node is written as *Node/na*. In the example figure, reaching node *N/bot* after moving up from *walk* causes a new goal node, the root node of tree T_2 *N/na* to be instantiated. The headcorner of an auxiliary tree is always the foot node. The auxiliary tree is then recursively ascended using the head-corner traversal until the root node *N/na* is reached. The root node of tree T_2 now spans the input string from $2, 4, 3, 4$ where the foot node spans $3, 4$. The goal is now completed and hence the parser executes a *completion* step and continues traversal in tree T_1 at node *N/top*. Once siblings have been recognized the traversal moves up to the parent of the headcorner node. The other kind of prediction is when a substitution node is reached during the traversal up to the root node. For example, the node *NP* in tree T_1 is a substitution node which introduces a goal node which is the root node *NP/top* of tree T_0. Traversal of tree T_0 occurs as usual. A *completion* step matches the root node of T_0 with the substitution node *NP* and the parser subsequently reaches the first goal node that was predicted: the root node *S/na* of tree T_1. The parser has then successfully parsed the input string *Ella took a walk.*

3.3 Data Structures

The parsing algorithm uses the following data structures. In the most general case, while parsing TAGs, the parser has to compute spans over the input string of the form i, j, fl, fr, where i, j is the span of the edge being processed and fl, fr are the spans of the foot node dominated by that edge. Each span of i, j over the input string of length len is stored in a two-dimensional array of size $len \times len$ called the `Chart`. Each entry in this array is a heap of *state*s which sorts each subtree span according to its probability. Each *state* is a 9-tuple defined as follows:

```
State = <n, goal, pos, fl, fr, type, hw, postag, {b, c, Pr}>
n:      node
goal:   goal node
pos:    {top, bot}
fl, fr: foot span
type:   {init, goal, item}
hw:     head word of node n's elementary tree
postag: part of speech tag of headword
b:      backpointer to State
c:      backpointer to State
Pr:     probability of State when backpointers are b,c
{b,c,Pr}:  backpointer list (blist); corresponds to the n-best State list
```

As new edges are added, they are stored in an agenda list called `Agenda` a list of items termed `Proc` which records which states were added to and the Chart entry where they were entered. Each entry in the chart is a heap of states. The formal definitions are as follows:

```
Proc  = <i, j, State>
Sheap = { s | s is State }
Agenda = { p | p is Proc }
Chart = A{len, len} with a_{i,j} = Sheap
```

3.4 Tree Traversal and Chart Maintenance Functions

The following description assumes that all the elementary trees in the grammar are binary branching. While explicit conversion to binary branching structures is usually avoided in CF parsing, in TAG parsing due to the use of adjunction there are far fewer cases of ternary or greater branching (we confirmed this on the XTAG English grammar and a Treebank grammar extracted from the Penn Treebank).

add ($\{i, j\}$, State) adds State to $\{i, j\}$ only if it does not exist. Also State.type = init is taken to be equal to goal for the equality test. If the State exists but the backpointers are not in the blist append the new backpointers to the blist. Use cmbprob(State,Pr) to update the state probability and sort blist to keep the most probable backpointers at head of list. Previous prob values can be stored with the blist to compute n-best values.

(State) exists in ($\{i, j\}$) operator that takes a State and checks chart at $\{i, j\}$ and returns Sheap or State of matches

anchors_for_tree(tree, index) takes a tree and an word index and returns anchor nodes of the tree if tree is lexicalized by word at index

init_rootnodes(label) takes a label and returns all initial trees rooted by nodes with that label

aux_rootnodes(label) takes a label and returns all auxiliary trees rooted by nodes with that label

is_adjoinable(node) returns true if an adjunction is possible at this node: rules out nodes like subst nodes, terminal nodes, footnodes, depending on the defn of adjunction used

is_subst(node) returns true if node is a subst node

headtype(State) if State.pos eq bot return ADJOIN else return context around node possible contexts = { `UNARY_HEAD, LEFT_HEAD, RIGHT_HEAD` }

nodetype(node) returns type of node. possible types = { `ANCHOR, TERMINAL, SUBST, EPS, FOOT` }

headcorner(node) returns distinguished node called headcorner. The headcorner for the rootnode of auxiliary trees is always the footnode; for initial trees it is the (leftmost) anchor. For nodes other than the rootnode the headcorner is picked recursively from the ranked list in the defn of nodetype(node).

comptype(node) completer type, one of the following:

`COMPL_INIT` : root of initial tree

`COMPL_AUX` : root of auxiliary tree

`COMPL_INTERNAL` : otherwise; goal was internal to a tree

prob(node, tree, node.lex, tree.lex) returns probability of adj/subst used for beam search

cmbprob(State, pr) if (Pr > State → Pr) then State → Pr = Pr

3.5 Parsing Algorithm

Parsing begins by predicting the root nodes which have the distinguished `TOP` label. The parser uses the type *init* to initialize the top-down prediction of the headcorner. The *init* edge after being added to the agenda is converted to a *goal* edge. Edges that are of the type *item* are edges that span some portion of the input string. The closure functions `init_head`, `move_up` and `completer` which recursively enter new edges into the Agenda. Parsing continues until there are no further edges to be processed in the Agenda. The closure function are detailed in subsequent sections.

```
Algorithm HParse:

__BEGIN__
```

```
N <- init_rootnodes(TOP)
start <- 0
len <- length(sent)+1
foreach n in N
     Agenda <- add({start,len}, <n, n, top, _, _, init, _, _>)

foreach p in Agenda
  {
     N1list <- init_head(p)
     N2list <- move_up(p)
     N3list <- completer(p)
     Agenda <- { N1list , N2list , N3list }
  }

get_derivations(TOP)

__END__
```

3.5.1 Initialize Headcorner

This function initializes the headcorner of each *init* edge that was inserted into the chart due to some top-down prediction by the parser. When the headcorner is an anchor or terminal symbol (`ANCHOR/TERMINAL`) or an empty element (denoted `EPS`), the relevant portion of the input string is scanned and a new edge which spans that portion is inserted into the Agenda. In the case of a substitution node (`SUBST`) each root node that could be substituted in that position with the appropriate span of the input is predicted as a new *goal* edge. Note that while we going bottom-up we can rule out any tree which cannot span the string to the left/right of the headcorner node. In the case of a foot node (`FOOT`) nothing special needs to be done apart from recording the span of the foot node since the function `move_up` will ensure the traversal to the root node of the auxiliary tree.

```
init_head (p)
{
  s <- p.State
  if (type(s.n) neq init) { return }

  switch(nodetype(hc <- headcorner(s.n))) {

  case ANCHOR/TERMINAL:
     for (k = p.i; k < p.j; k++)
       N <- anchors_for_tree(tree(hc), k)
       foreach n in N
          Agenda <- add({k,k+1}, <n, s.goal, bot, _, _, item, _, _>)

  case EPS:
     if (p.i eq p.j)
        Agenda <- add({p.i,p.j}, <s.n, s.goal, top, _, _, item, _, _>)

  case SUBST:
     N <- init_rootnodes(label(hc))
     foreach n in N
        if (b <- <n, n, top, _, _, item, _, _> exists in {p.i, p.j})
           Pr <- prob(hc, tree(b.n), s.lex, b.lex)
           Agenda <- add({p.i, p.j}, <hc, s.goal, top, _, _, item, b, 0>)
        else
           Agenda <- add({p.i, p.j}, <hc, s.goal, top, _, _, goal, _, _>)
           Agenda <- add({p.i, p.j}, <n, n, top, _, _, init, _, _>)

  case FOOT:
     Agenda <- add({s.fl, s.fr}, <hc, s.goal, bot, s.fl, s.fr, item, _, _>)
  }
  s.type <- goal
  return(Agenda)
}
```

3.5.2 Move Up

The function `move_up` proceeds upwards from each *item* edge towards the *goal* edges. In this traversal, new *goal* edges can be proposed. If the edge is at the bottom of a node *Node/bot*, then adjunction is predicted at that node. The adjunction might not be successful and so the top of the node *Node/top* is also predicted (corresponding to a null adjunction).

The other cases cover the traversal of sibling nodes where new goal edges are inserted into the Agenda and the case of unary nodes where the parent is introduced as a new *item* edge.

```
move_up (p)
{
  s <- p.State
  g <- s.goal
  if (s.type neq item) OR ((s.pos eq top) AND (s.n eq g)) { return }
  pt <- parent(s.n)
  switch(headtype(s)) {

  case ADJOIN:
     Agenda <- add({p.i,p.j}, <s.n, g, top, s.fl, s.fr, item, s, _>)

     if is_adjoinable(s.n)
        N <- aux_rootnodes(label(s.n))
        for (i = start; i <= p.i; i++)
```

```
            for (j = p.j; j < len; j++)
                if (i eq p.i) AND (j eq p.j) { continue }
                foreach n in N
                   if (b <- <n, n, top, p.i, p.j, item, _, _> exists in {i,j})
                      Pr <- prob(s.n, tree(b.n), s.lex, b.lex)
                      Agenda <- add({i,j}, <s.n, g, top, p.i, p.j, item, b, s>)
                   else
                      Agenda <- add({i,j}, <n, n, top, p.i, p.j, init, _, _>)

  case UNARY_HEAD:
     Agenda <- add({p.i, p.j}, <pt, g, bot, s.fl, s.fr, item, s, _>)

  case LEFT_HEAD:
     r <- rightnode(s.n)
     for (k = p.j; k < len; k++)
         if (b <- <r, r, top, _, _, item, _> exists in {p.j, k})
            Agenda <- add({p.i, k}, <pt, g, bot, s.fl, s.fr, item, s, b>)
         else
            Agenda <- add({p.j, k}, <r, r, top, _, _, init, _, _>)

  case RIGHT_HEAD:
     l <- leftnode(s.n)
     for (k = start; k <= p.i; k++)
         if (b <- <l, l, top, _, _, item, _> exists in {k, p.i})
            Agenda <- add({k, p.j}, <pt, g, bot, s.fl, s.fr, item, s, b>)
         else
            Agenda <- add({k, p.i}, <l, l, top, _, _, init, _, _>)
  }
  return(Agenda)
}
```

3.5.3 Completer

This is the most complex function because it is responsible for finally recognizing that an adjunction or substitution has occurred. The root node of an initial tree (COMPL_INIT) or an auxiliary tree (COMPL_AUX) is matched up with the predicted *goal* edge and in the case of adjunction whether the span of the foot node matches the span of the node where adjunction was predicted. It is this step that is responsible for the worst case complexity of n^6 for TAG parsing as six independent indices into the input string have to be checked.

The completer also takes successfully recognized sibling nodes of headcorner nodes and then inserts new *item* edges for the parent of the headcorner.

```
completer (p)
{
  s <- p.State
  if (s.type neq item) OR (s.pos neq top) OR (s.n neq s.goal) { return }
  switch(comptype(s.n)) {

  case COMPL_INIT:
    Slist <- { <n, ?goal, top, _, _, goal, _> |
                    label(s.n) eq label(n),
                    is_subst(n) } exists in {p.i, p.j}
    for b in Slist
        Pr <- prob(s.n, tree(b.n), s.lex, b.lex)
        Agenda <- add({p.i, p.j}, <n, b.goal, top, _, _, item, s, 0>)

  case COMPL_AUX:
    Slist <- { <n, ?n.goal, bot, ?fl, ?fr, item, _> |
                    <footnode(tree(s.n)), _, bot, fl, fr, item, _> in {fl,fr},
                    label(s.n) eq label(n),
                  is_adjoinable(n) } exists in {s.fl, s.fr}
    for b in Slist
        Pr <- prob(s.n, tree(b.n), s.lex, b.lex)
        Agenda <- add({p.i, p.j}, <n, b.goal, top, b.fl, b.fr, item, b, s>)

  case COMPL_INTERNAL:
    pt <- parent(s.n)
    switch(headtype(s.n)) {

    case LEFT_HEAD:
      r <- rightnode(s.n)
      for (k = p.j; k < len; k++)
          Slist <- { <r, ?goal, top, ?fl, ?fr, item, _> } exists in {p.j, k}
          for b in Slist
              Agenda <- add({p.i, k}, <pt, b.goal, bot, b.fl, b.fr, item, b, s>)

    case RIGHT_HEAD:
      l <- leftnode(s.n)
      for (k = start; k <= p.i; k++)
          Slist <- { <l, ?goal, top, ?fl, ?fr, item, _> } exists in {k, p.i}
          for b in Slist
              Agenda <- add({k, p.j}, <pt, b.goal, bot, b.fl, b.fr, item, b, s>)
    }
  }
  return(Agenda)
}
```

3.6 Parser Implementation

The parser is implemented in ANSI C and runs on SunOS 5.x and Linux 2.x. Apart from the Treebank Grammar used in this chapter, the parser has been tested with the XTAG English Grammar.

The implementation optimizes for space at the expense of speed, e.g. the recognition chart is implemented as a sparse array thus taking considerably less than the worst case n^4 space and the lexical database is read dynamically from a disk-based hash table. For each input sentence, the parser produces as output a shared derivation forest which is a compact representation of all the derivation trees for that sentence. We use the definition of derivation forests for TAGs represented as CFGs, taking $O(n^4)$ space as defined in [Vijay-Shanker and Weir, 1993, Lang, 1994].

3.6.1 Conclusion

In this chapter, we described an implementation of a chart-based head-corner parser for LTAGs. In Appendix A we describe an experiment using this parser where we ran the parser on 2250 sentences from the Wall Street Journal and measured its observed time complexity.

Chapter 4

Applying Co-Training methods to Statistical Parsing

In this chapter we propose a new approach for training a statistical parser that uses a Co-Training method. The algorithm takes as input a small corpus ($\approx$ 9K sentences) annotated with parse trees, a large pool of unlabeled text and a tag dictionary containing each word in this training set. The algorithm iteratively labels the unlabeled data set with parse trees. In experimental comparisions with a model that only uses labeled data we find an increase in accuracy when we combine labeled and unlabeled data using our unsupervised method.

4.1 Introduction

The current crop of statistical parsers share a similar training methodology. They train from the Penn Treebank [Marcus et al., 1993]; a collection of 40,000 sentences that are labeled with corrected parse trees (approximately a million word tokens). We would like to explore in this chapter methods for statistical parsing that can be used to combine small amounts of labeled data with unlimited amounts of unlabeled data. In the experiment reported here, we use around 9,000 sentences of bracketed data. Such methods are attractive for the following reasons:

- Bracketing sentences is an expensive process. A parser that can be trained on a small amount of labeled data will reduce this annotation cost.

- Creating statistical parsers for novel domains and new languages will become easier.
- Combining labeled data with unlabeled data allows exploration of unsupervised methods which can now be tested using evaluations compatible with supervised statistical parsing.

In this chapter we introduce a new approach that combines unlabeled data with a small amount of labeled (bracketed) data to train a statistical parser. We use a Co-Training method [Yarowsky, 1995, Blum and Mitchell, 1998, Goldman and Zhou, 2000] that has been used previously to train classifiers in applications like word-sense disambiguation [Yarowsky, 1995], document classification [Blum and Mitchell, 1998] and named-entity recognition [Collins and Singer, 1999] and apply this method to the more complex domain of statistical parsing.

4.2 Co-Training methods for parsing

Many supervised methods of learning from a Treebank have been studied. The question we want to pursue in this chapter is whether unlabeled data can be used to improve the performance of a statistical parser and at the same time reduce the amount of labeled training data necessary for good performance. We will assume the data that is input to our method will have the following characteristics:

1. A small set of sentences labeled with corrected parse trees.
2. A pair of probabilistic models that form parts of a statistical parser. This pair of models must be able to mutually constrain each other.
3. A tag dictionary (used within a backoff smoothing strategy) for labels are not covered in the labeled set.

The pair of probabilistic models can be exploited to bootstrap new information from unlabeled data. Since both of these steps ultimately have to agree with each other, we can utilize an iterative method called Co-Training that attempts to increase agreement between a pair of statistical models by exploiting mutual constraints between their output.

Co-Training has been used before in applications like word-sense disambiguation [Yarowsky, 1995], web-page classification [Blum and Mitchell, 1998] and named-entity identification [Collins and Singer, 1999 In all of these cases, using unlabeled data has resulted in performance that rivals training solely from labeled data. However, these previous approaches were on tasks that involved identifying the right label from a small set of labels (typically 2), and in a relatively small parameter space. Compared to these earlier models, a statistical parser has a very large parameter space and the labels that are expected as output are parse trees which have to be built up recursively. We discuss previous work in combining labeled and unlabeled data in more detail in Section 4.6.

4.2.1 Lexicalized Grammars: Stochastic Tree Adjoining Grammars

Our approach exploits the lexicalization of structure in order to bootstrap new information from unlabeled data. We use the formalism of Lexicalized Tree Adjoining Grammar (LTAG) [Joshi and Schabes, 1992] in our experiments because it offers a simple parameterization allowing us to experiment with unsupervised methods while offering the capability to model aspects such as subcategorization and argument/adjunct distinctions which have to be explicitly handled in a stochastic lexicalized context-free grammar. This convenience comes with the price of having to handle carefully sparse data problems that arise as a result of having larger elementary structures.

We explain the basic model for probabilistic parsing using an LTAG by using an example (the details of the model are discussed in Section 4.3, while details of the LTAG formalism are given in [Joshi and Schabes, 1992]).

Consider an example sentence: *Ms. Haag plays Elianti.* In an LTAG, each word in this sentence is associated with a set of *elementary trees.* There are two kinds of elementary trees. Certain trees with distinguished non-terminal node (termed as footnode) in the frontier of the tree can recursively embed into other trees. These are called *auxiliary trees.* The non-recursive trees are called *initial trees.* The word that selects a tree is called the *anchor* of that tree. In Figure 4.1 the set of trees anchored by each word in this sentence is shown. Parsing involves the attachment of nodes in each of these trees into the appropriate node in another tree. These attachments are categorized into two operations in the formalism. For example, the parse for the example sentence shown in Figure 4.2 is constructed using the following steps. In Figure 4.1, the tree **1** is an auxiliary tree that replaces the rootnode of tree **2** with its root and foot node: this is termed as adjunction. This operation forms the constituent *Ms. Haag* in Figure 4.2. Replacing the $NParg$ nodes in the frontier of the tree **3** for *plays* with the trees **2** and **4** is termed as substitution. This completes the parse of the sentence resulting in the parse tree shown in Figure 4.2.

In this chapter, we will characterize both the operations of substitution and adjunction simply as one term: **attachment**.

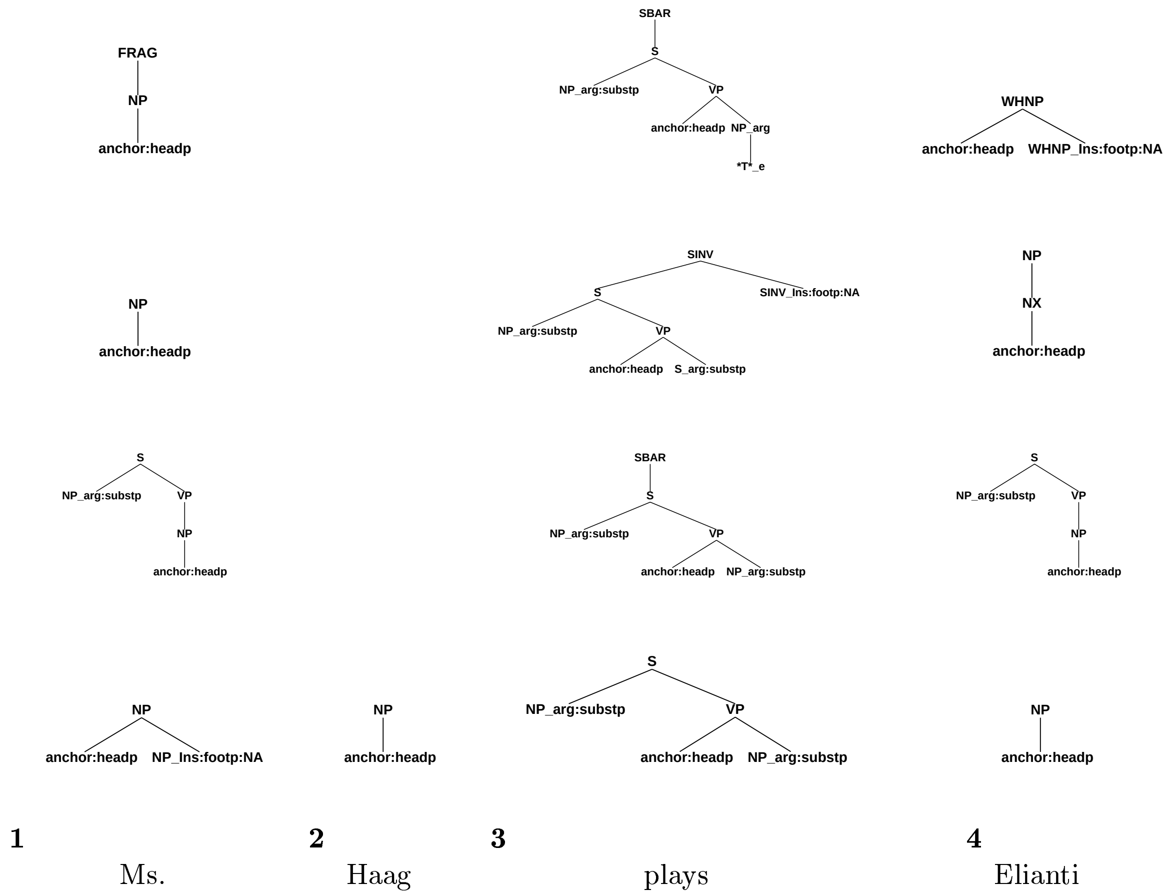


Figure 4.1: Parsing of an example sentence: *Ms. Haag plays Elianti.*

A stochastic LTAG derivation proceeds as follows [Schabes, 1992, Resnik, 1992]. An initial tree is selected with probability P_{INIT} and subsequent substitutions and adjunctions are performed each with probability P_{ATTACH}.

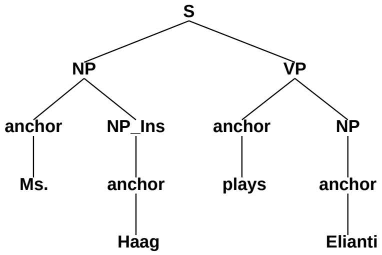


Figure 4.2: Parse tree produced by finding a consistent set of attachments that covers the sentence.

For each τ that can be valid start of a derivation:

$$\sum_{\tau} P_{\text{INIT}}(\tau) = 1$$

Each subsequent attachment occurs independently, with the condition that:

$$\sum_{\tau'} P_{\text{ATTACH}}(\tau' \rightarrow \tau, \eta) = 1$$

where, τ' is substituting or adjoining into node η in tree τ. For adjunctions there is an additional factor which is required for the probability to be well-formed:

$$P_{\text{ATTACH}}(\textsc{na} \rightarrow \tau, \eta)$$

which is the probability that there is no adjunction (NA) at node η in τ.

Each LTAG derivation $\mathcal{D}$ which was built starting from tree α with n subsequent attachments has the probability:

$$Pr(\mathcal{D}) = P_{\text{INIT}}(\alpha) \prod_{1 \leq i \leq n} P_{\text{ATTACH}}(\tau_i' \rightarrow \tau, \eta)$$

This derivation $\mathcal{D}$ can be drawn graphically as a tree where each node in this derivation tree is an elementary tree in the original LTAG. For example, combining the lowermost trees for each word in Figure 4.1 gives us the derivation tree in Figure 4.3. By combining each of the elementary trees we can also draw the usual constituent structure shown in

Figure 4.2. This structure can be further simplified by removing certain nodes that are in the constituent structure as a result of the adjunction operation. This final parse is shown in Figure 4.4.

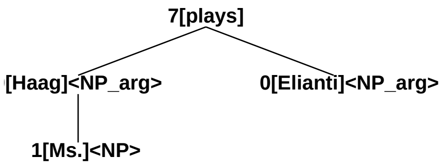


Figure 4.3: A derivation indicating all the attachments between trees that have occurred during the parse of the sentence.

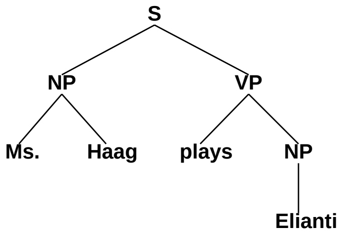


Figure 4.4: Parse tree after all nodes that were inserted for parsing are removed.

4.2.2 Lexicalized Grammars and Mutual Constraints

Using fully lexicalized grammars allows us to define the parsing problem in terms of two mutually constraining processes. Parsing using a lexicalized grammar is done in two steps:

1. Assigning a set of lexicalized structures to each word in the input sentence (as shown in Figure 4.1).

2. Finding the correct attachments between these structures to get the best parse (as shown in Figure 4.4).

Each of these two steps involves ambiguity which can be resolved using a statistical model. By explicitly representing these two steps independently, we can pursue independent statistical models for each step:

1. Each word in the sentence can take many different lexicalized structures. We can introduce a statistical model that disambiguates the lexicalized structure assigned to a word depending on the local context.

2. After each word is assigned a certain set of lexicalized structures, finding the right parse tree involves computing the correct attachments between these lexicalized structures. Disambiguating attachments correctly using an appropriate statistical model is essential to finding the right parse tree.

These two models have to agree with each other on the trees assigned to each word in the sentence. Not only do the right trees have to be assigned as predicted by the first model, but they also have to fit together to cover the entire sentence as prediced by the second model. This represents the mutual constraint that each model places on the other.

4.2.3 Tag Dictionaries

For the words that appear in the (unlabeled) training data, we collect a list of part-of-speech labels and trees that each word is known to select in the Treebank. This information is stored in a POS tag dictionary and a tree dictionary. It is important to note that no frequency or any other distributional information is stored. The only information stored in the dictionary is which tags or trees can be selected by each word in the training data.

The problem of lexical coverage is a severe one for unsupervised approaches. The use of tag dictionaries is a way around this problem. Such an approach has already been used for unsupervised part-of-speech tagging in [Brill, 1997] where seed data of which POS tags can be selected by each word is given as input to the unsupervised tagger.

Without such lexical knowledge the learning algorithm can easily lose its way in the search space. This was reported in [Elworthy, 1994] which studied carefully the problem of learning a part-of-speech tagger by combining labeled and unlabeled data using HMMs. Previously [Cutting et al., 1992] had shown that by constraining the HMM with some prior knowledge by hand, very high performance could be achieved in unsupervised part-of-speech tagging.

In future work, it would be interesting to extend models for unknown-word handling to the creation of such tag dictionaries. But for this chapter, we take such dictionaries as granted.

4.3 Models

As described before, we treat parsing as a two-step process. The two models that we use are:

1. H1: selects trees based on previous context (tagging probability model)
2. H2: computes attachments between trees and returns best parse (parsing probability model)

4.3.1 H1: Tagging probability model

We select the most likely trees for each word by examining the local context. The statistical model we use to decide this is the trigram model that was used by B. Srinivas in his SuperTagging model [Srinivas, 1997a]. The model assigns an n-best lattice of tree assignments associated with the input sentence with each path corresponding to an assignment of an elementary tree for each word in the sentence. (for further details, see [Srinivas, 1997a]).

$$P(\mathbf{T}|\mathbf{W})$$

$$= P(T_0 \ldots T_n | W_0 \ldots W_n) \tag{4.1}$$

$$= \frac{P(T_0 \ldots T_n) \times P(W_0 \ldots W_n | T_0 \ldots T_n)}{P(W_0 \ldots W_n)} \tag{4.2}$$

$$\approx P(T_i | T_{i-2} T_{i-1}) \times P(W_i | T_i) \tag{4.3}$$

where, $T_0 \ldots T_n$ is a sequence of elementary trees assigned to the sentence $W_0 \ldots W_n$.

We get (4.2) by using Bayes theorem and we obtain (4.3) from (4.2) by ignore the denominator and by applying the usual Markov assumptions.

The output of this model is a probabilistic ranking of trees for the input sentence which is sensitive to a small local context window. The tagging model also incorporates the tag and tree dictionary that was described in Section 4.2.3. It uses the dictionary to seed previously unseen trees to the parsing model with a default backoff value that is obtained by discounting low counts in the tagger model.

4.3.2 H2: Parsing probability model

Once the words in a sentence have selected a set of elementary trees, parsing is the process of attaching these trees together to give us a consistent bracketing of the sentences. Notation: Let τ stand for an elementary tree which is lexicalized by a word: w and a part of speech tag: p.

Let P_{INIT} (introduced earlier in 4.2.1) stand for the probability of being root of a derivation tree defined as follows:

$$\begin{aligned}
&\Pr(\tau, w, p \mid top = 1) = \\
&\quad \Pr(\tau \mid top = 1) \times && (4.4)\\
&\quad \Pr(p \mid \tau, top = 1) \times && (4.5)\\
&\quad \Pr(w \mid \tau, p, top = 1); && (4.6)
\end{aligned}$$

where, the variable top indicates that τ is the tree that begins the current derivation. However, there is a useful approximation for P_{INIT}:

$$\Pr(\tau, w, p \mid top = 1) \approx \Pr(top = 1 \mid label)$$

where, $label$ is the label of the root node of τ.

$$\begin{aligned}
&\hat{\Pr}(top = 1 \mid label) = \\
&\quad \frac{Count(top = 1, label) + \alpha}{Count(top = 1) + N\alpha} && (4.7)
\end{aligned}$$

where N = number of bracketing labels.

Let $P_{\textsc{Attach}}$ (introduced earlier in 4.2.1) stand for the probability of attachment of τ' into another τ:

$$\Pr(\tau', p', w' | Node, \tau, w, p) \tag{4.8}$$

$$\Pr(\textsc{na} | Node, \tau, w, p) \tag{4.9}$$

(4.9) is estimated from the training data using the following formula. Note that (4.9) will be 0 for substitution attachments.

$$\begin{aligned} &Count(Node, \tau, w, p, \textsc{na}) = \\ &\quad Count(\tau, w, p) - & (4.10) \\ &\quad Count(Node, \tau, w, p) & (4.11) \end{aligned}$$

Eqn (4.10) is the total number of times we have seen τ, w, p and Eqn (4.11) is the number of times node *Node* in tree τ, w, p has been seen as being modified. This gives us the value of NA at node *Node*.

We decompose (4.8) into the following components:

$$\begin{aligned} &\Pr(\tau', p', w' | Node, \tau, w, p) = \\ &\quad \Pr(\tau' | Node, \tau, w, p) \times & (4.12) \\ &\quad \Pr(p' | \tau', Node, \tau, w, p) \times & (4.13) \\ &\quad \Pr(w' | p', \tau', Node, \tau, w, p); & (4.14) \end{aligned}$$

We do a similar decomposition for (4.9).

For each of the equations above, we use a backoff model which is used to handle sparse data problems. We compute a backoff model as follows:

Let e_1 stand for the original lexicalized model and e_2 be the backoff level which only uses part of speech information:

e_1: $Node, \tau, w, p$

e_2: $Node, \tau, p$

For both P_{INIT} and P_{ATTACH}, let $Count(e_1) = c$. Then the backoff model is computed as follows:

$$\lambda(c)e_1 + (1 - \lambda(c))e_2$$

where, $\lambda(c) = \frac{c}{(c+D)}$ and where, D is the diversity of e_1, the number of distinct counts for e_1.

For P_{ATTACH} we further smooth probabilities (4.12), (4.13) and (4.14). We use (4.12) as an example, the other two are handled in the same way.

$$\begin{aligned}\hat{\Pr}(\tau' | Node, \tau, w, p) = \\ \frac{(Count(Node, \tau, w, p, \tau') + \alpha)}{(Count(Node, \tau, w, p) + k\alpha)}\end{aligned} \tag{4.15}$$

$$\begin{aligned}Count(Node, \tau, w, p) = \\ \sum_y Count(Node, \tau, w, p, y)\end{aligned} \tag{4.16}$$

where k is the diversity of adjunction, that is: the number of different trees that can attach at that node.

For our experiments, the value of α is set to $\frac{1}{100{,}000}$.

4.4 Co-Training algorithm

We are now in the position to describe the Co-Training algorithm, which combines the models described in Section 4.3.1 and in Section 4.3.2 in order to iteratively label a large pool of unlabeled data.

We use the following datasets in the algorithm:

labeled a set of sentences bracketed with the correct parse trees.

cache a small pool of sentences which is the focus of each iteration of the Co-Training algorithm.

unlabeled a large set of unlabeled sentences. The only information we collect from this set of sentences is a tree-dictionary: *tree-dict* and part-of-speech dictionary: *pos-dict*. Construction of these dictionaries is covered in Section 4.2.3.

In addition to the above datasets, we also use the usual development test set (termed *dev* in this chapter), and a test set (called *test*) which is used to evaluate the bracketing accuracy of the parser.

The Co-Training algorithm consists of the following steps which are repeated iteratively until all the sentences in the set *unlabeled* are exhausted.

1. Input: *labeled*
2. Update cache:
   - If *unlabeled* is empty: exit.
   - Randomly select sentences from *unlabeled* and refill the *cache*.
3. Train models H1 and H2 using *labeled*
4. Apply H1 to *cache*
5. Apply H2 to output of Step 4.
6. Pick best n from output of Step 5.
7. Remove best n from *cache* and append new labeled data to *labeled*
8. n = 2n; Go to Step 2.

For this expt, n = 10, run for 12 iterations (covering 20480 of the sentences in *unlabeled*) and then add the best parses for all the remaining sentences.

4.5 Experiment

The experiments we report were done on the Penn Treebank WSJ Corpus [Marcus et al., 1993]. The size of the set *labeled* is 9625 sentences which is Sections 02-06. The set *unlabeled* of sentences for training the model (without any annotation) was 30137 sentences (also taken from the Treebank for convenience with their structures omitted). We set the size of the cache to a maximum value of 3000 sentences in this chapter. We initially set the cache to a low value and increase it during later iterations.

While it might seem expensive to run the parser over the cache multiple times, we use the pruning capabilities of the parser to good use here. During the iterations we set the beam size to a value which is likely to prune out all derivations for a large portion of the cache except the most likely ones. This also causes the parser to run faster, hence avoiding the usual problem with running an iterative algorithm over thousands of sentences. In the initial runs we also limit the length of the sentences entered into the cache since they are more likely to beat out the longer sentences in any case. The beam size is reset when running the parser on the test data to allow the parser a better chance at finding the most likely parse.

We scored the output of the parser on Section 0 of the Wall Street Journal Penn Treebank. The following are some aspects of the scoring that might be useful for comparision with other results.

- No punctuations are scored, including sentence final punctuation.
- Empty elements are not scored.
- used EVALB (written by Satoshi Sekine and Michael Collins) for evaluation; with the standard parameter file.
- Also, we used Adwait Ratnaparkhi's part-of-speech tagger [Ratnaparkhi, 1996] to tag unknown words in the test data.

We obtained 81.2% and 78.94% labeled bracketing precision and recall respectively. The baseline model which was only trained on the initial 9K sentences of labeled data performed at 67.43% and 64.93% precision and recall. Thus, we can see an improvement in accuracy as a result of Co-Training. I am currently running more experiments and will report the latest results during the proposal defense.

Part of the reason for the low accuracy overall is that smoothing for unknown words is not very sophisticated especially when novel trees are seen in the test data (this occurs very rarely) or when trees seen before while training occur with words not previously associated with them (this is the main cause of the lower bracketing accuracy as compared to the supervised results).

For the final version of this chapter we plan to present the results of an experiment that is currently in progress which uses a larger set of unlabeled data (a set of 23M words of Wall Street Journal text) and which trains the initial models using the entire Treebank. This experiment involves a greater number of iterations of Co-Training and is not as greedy as our current setting.

4.6 Previous Work: Combining Labeled and Unlabeled Data

The two-step procedure used in our Co-Training method for statistical parsing was incipient in the SuperTagger [Srinivas, 1997a] which is a statistical model for tagging sentences with elementary lexicalized structures. This was particularly so in the Lightweight Dependency Analyzer (LDA), which used shortest attachment heuristics after an initial SuperTagging stage to find syntactic dependencies between words in a sentence. However, there was no statistical model for attachments and the notion of mutual constraints between these two steps was not exploited in this work.

Previous studies in unsupervised methods for parsing have concentrated on the use of inside-outside algorithm [Lari and Young, 1990, Carroll and Rooth, 1998a]. However, there are several limitations of the inside-outside algorithm for unsupervised parsing, see [Marcken, 1995] for some experiments that draw out the mismatch between minimizing error rate and iteratively increasing the likelihood of the corpus. Other approaches have tried to move away from phrase structural representations into dependency style parsing [Lafferty et al., 1992, Fong and Wu, 1996]. However, there are inherent computational limitations due to the vast search space (see [Pietra et al., 1994] for discussion).

[Chelba and Jelinek, 1998] combine unlabeled and labeled data for parsing with a view towards language modeling applications. The goal here is not to get the right bracketing or dependencies but to reduce the word error rate in a speech recognizer.

Our approach is closely related to previous Co-Training methods [Yarowsky, 1995, Blum and Mitchell, 1998, Goldman and Zhou, 2000, Collins and Singer, 1999]. [Yarowsky, 1995] first introduced an iterative method for increasing a small set of seed data used to disambiguate dual word senses by exploiting the constraint that in a discourse only one sense of a word is used. This use of unlabeled data improved performance of the disambiguator above that of purely supervised methods. [Blum and Mitchell, 1998] further embellished this approach and gave it the name of Co-Training. Their definition of Co-Training includes the notion (exploited in this chapter) that different models can constrain each other by exploiting different 'views' of the data. They also prove some PAC results on learnability. They also discuss an application of classifying web pages by using their method of mutually constrained models. [Collins and Singer, 1999] further extends the use of classifiers that have mutual constraints by adding terms to AdaBoost which force the classifiers to agree (called Co-Boosting). [Goldman and Zhou, 2000] provide a variant of Co-Training which is suited to the learning of decision trees where the data is split up into different equivalence classes for each of the models and they use hypothesis testing to determine the agreement between the models. In future work we would like to experiment whether some of these ideas could be incorporated into our model.

4.7 Conclusion

In this chapter, we proposed a new approach for training a statistical parser that combines labeled with unlabeled data. It uses a Co-Training method where a pair of models attempt to increase their agreement on labeling the data. The algorithm takes as input a small corpus ($\approx$ 9K sentences) annotated with parse trees, a large pool of unlabeled text and a dictionary of possible lexicalized structures for each word in this training set. The algorithm iteratively labels the unlabeled data set with parse trees. We obtained 56.2%

and 51.94% labeled bracketing precision and recall respectively. The baseline model which was only trained on the initial 9K sentences of labeled data performed at 37.43% and 31.93% precision and recall. Thus, we find an increase in accuracy over using only labeled data when we combine labeled and unlabeled data using our unsupervised method.

Chapter 5

Prefix Probabilities from Probabilistic TAGs

Language models for speech recognition typically use a probability model of the form $\Pr(a_n | a_1, a_2, \ldots, a_{n-1})$. Probabilistic grammars, on the other hand, are typically used to assign structure to utterances. A language model of the above form is constructed from such grammars by computing the prefix probability $\sum_{w \in \Sigma^*} \Pr(a_1 \cdots a_n w)$, where w represents all possible terminations of the prefix $a_1 \cdots a_n$. The main result in this chapter is an algorithm to compute such prefix probabilities given a probabilistic Tree Adjoining Grammar (TAG). The algorithm achieves the required computation in $\mathcal{O}(n^6)$ time. The probability of subderivations that do not derive any words in the prefix, but contribute structurally to its derivation, are precomputed to achieve termination. This algorithm enables existing corpus-based estimation techniques for probabilistic TAGs to be used for language modeling.

5.1 Prefix Probabilities

Given some word sequence $a_1 \cdots a_{n-1}$, speech recognition language models are used to hypothesize the next word a_n, which could be any word from the vocabulary Σ. This is typically done using a probability model $\Pr(a_n | a_1, \ldots, a_{n-1})$. Based on the assumption that modeling the hidden structure of natural language would improve performance of such language models, some researchers tried to use probabilistic context-free grammars (CFGs) to produce language models [Wright and Wrigley, 1989, Jelinek and Lafferty, 1991, Stolcke, 1995]. The probability model used for a probabilistic grammar was $\sum_{w \in \Sigma^*} \Pr(a_1 \cdots a_n w)$. However, language models that are based on trigram probability models out-perform probabilistic CFGs. The common wisdom about this failure of CFGs is that trigram models are lexicalized models while CFGs are not.

Tree Adjoining Grammars (TAGs) are important in this respect since they are easily lexicalized while capturing the constituent structure of language. More importantly, TAGs allow greater linguistic expressiveness. The trees associated with words can be used to encode argument and adjunct relations in various syntactic environments. TAGs have been shown to have relations with both phrase-structure grammars and dependency grammars [Rambow and Joshi, 1995], which is relevant because recent work on *structured* language models [Chelba et al., 1997] have used dependency grammars to exploit their lexicalization. We use probabilistic TAGs as such a *structured* language model in contrast with earlier work where TAGs have been exploited in a class-based n-gram language model [Srinivas, 1996].

This chapter derives an algorithm to compute prefix probabilities $\sum_{w \in \Sigma^*} \Pr(a_1 \cdots a_n w)$. The algorithm assumes as input a probabilistic TAG G and a string which is a prefix of some string in $L(G)$, the language generated by G. This algorithm enables existing corpus-based estimation techniques [Schabes, 1992] in probabilistic TAGs to be used for language modeling.

5.2 Notation

A probabilistic Tree Adjoining Grammar (STAG) is represented by a tuple $(N, \Sigma, \mathcal{I}, \mathcal{A}, \phi)$ where N is a set of nonterminal symbols, Σ is a set of terminal symbols, $\mathcal{I}$ is a set of **initial** trees and $\mathcal{A}$ is a set of **auxiliary** trees. Trees in $\mathcal{I} \cup \mathcal{A}$ are also called **elementary** trees.

We refer to the root of an elementary tree t as R_t. Each auxiliary tree has exactly one distinguished leaf, which is called the **foot**. We refer to the foot of an auxiliary tree t as F_t. We let V denote the set of all nodes in the elementary trees.

For each leaf N in an elementary tree, except when it is a foot, we define $\mathit{label}(N)$ to be the label of the node, which is either a terminal from Σ or the empty string ϵ. For each other node N, $\mathit{label}(N)$ is an element from N.

At a node N in a tree such that $\mathit{label}(N) \in N$ an operation called **adjunction** can be applied, which excises the tree at N and inserts an auxiliary tree.

Function ϕ assigns a probability to each adjunction. The probability of adjunction of $t \in \mathcal{A}$ at node N is denoted by $\phi(t, N)$. The probability that at N no adjunction is applied is denoted by $\phi(\mathbf{nil}, N)$. We assume that each STAG G that we consider is **proper**. That is, for each N such that $\mathit{label}(N) \in N$,

$$\sum_{t \in \mathcal{A} \cup \{\mathbf{nil}\}} \phi(t, N) = 1.$$

For each non-leaf node N we construct the string $\mathit{cdn}(N) = \widehat{N_1} \cdots \widehat{N_m}$ from the (ordered) list of children nodes $N_1, \ldots, N_m$ by defining, for each d such that $1 \leq d \leq m$, $\widehat{N_d} = \mathit{label}(N_d)$ in case $\mathit{label}(N_d) \in \Sigma \cup \{\epsilon\}$, and $\widehat{N_d} = N_d$ otherwise. In other words, children nodes are replaced by their labels unless the labels are nonterminal symbols.

To simplify the exposition, we assume an additional node for each auxiliary tree t, which we denote by $\perp$. This is the unique child of the actual foot node F_t. That is, we change the definition of cdn such that $\mathit{cdn}(F_t) = \perp$ for each auxiliary tree t. We set

$$V^{\perp} = \{N \in V \mid \mathit{label}(N) \in N\} \cup \Sigma \cup \{\perp\}.$$

We use symbols $a, b, c, \ldots$ to range over Σ, symbols $v, w, x, \ldots$ to range over Σ^*, symbols $N, M, \ldots$ to range over $V^{\perp}$, and symbols $\alpha, \beta, \gamma, \ldots$ to range over $(V^{\perp})^*$. We use $t, t', \ldots$ to denote trees in $\mathcal{I} \cup \mathcal{A}$ or subtrees thereof.

We define the predicate dft on elements from $V^{\perp}$ as $\mathit{dft}(N)$ if and only if (i) $N \in V$ and N dominates $\perp$, or (ii) $N = \perp$. We extend dft to strings of the form $N_1 \ldots N_m \in (V^{\perp})^*$ by defining $\mathit{dft}(N_1 \ldots N_m)$ if and only if there is a d $(1 \leq d \leq m)$ such that $\mathit{dft}(N_d)$.

For some logical expression p, we define $\delta(p) = 1$ iff p is true, $\delta(p) = 0$ otherwise.

5.3 Overview

The approach we adopt in the next section to derive a method for the computation of prefix probabilities for TAGs is based on transformations of equations. Here we informally discuss the general ideas underlying equation transformations.

Let $w = a_1 a_2 \cdots a_n \in \Sigma^*$ be a string and let $N \in V^{\perp}$. We use the following representation which is standard in tabular methods for TAG parsing. An **item** is a tuple $[N, i, j, f_1, f_2]$ representing the set of all trees t such that (i) t is a subtree rooted at N of some derived elementary tree; and (ii) t's root spans from position i to position j in w, t's foot node spans from position f_1 to position f_2 in w. In case N does not dominate the foot, we set $f_1 = f_2 = -$. We generalize in the obvious way to items $[t, i, j, f_1, f_2]$, where t is an elementary tree, and $[\alpha, i, j, f_1, f_2]$, where $cdn(N) = \alpha\beta$ for some N and β.

To introduce our approach, let us start with some considerations concerning the TAG parsing problem. When parsing w with a TAG G, one usually composes items in order to construct new items spanning a larger portion of the input string. Assume there are instances of auxiliary trees t and t' in G, where the yield of t', apart from its foot, is the empty string. If $\phi(t, N) > 0$ for some node N on the spine of t', and we have recognized an item $[R_t, i, j, f_1, f_2]$, then we may adjoin t at N and hence deduce the existence of an item $[R_{t'}, i, j, f_1, f_2]$ (see Fig. 5.1(a)). Similarly, if t can be adjoined at a node N to the left of the spine of t' and $f_1 = f_2$, we may deduce the existence of an item $[R_{t'}, i, j, j, j]$ (see Fig. 5.1(b)). Importantly, one or more other auxiliary trees with empty yield could wrap the tree t' before t adjoins. Adjunctions in this situation are potentially nonterminating.

One may argue that situations where auxiliary trees have empty yield do not occur in practice, and are even by definition excluded in the case of lexicalized TAGs. However, in the computation of the prefix probability we must take into account trees with non-empty yield which behave like trees with empty yield because their lexical nodes fall to the right of the right boundary of the prefix string. For example, the two cases previously considered in Fig. 5.1 now generalize to those in Fig. 5.2.

To derive a method for the computation of prefix probabilities, we give some simple recursive equations. Each equation *decomposes* an item into other items in all possible ways, in the sense that it expresses the probability of that item as a function of the

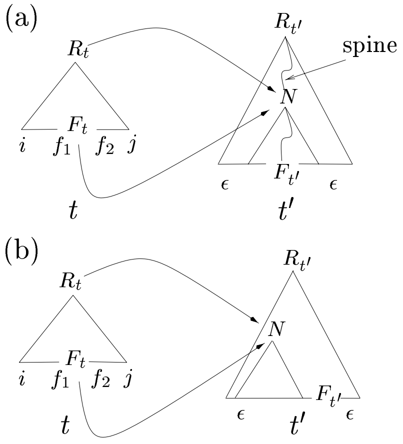


Figure 5.1: Wrapping in auxiliary trees with empty yield

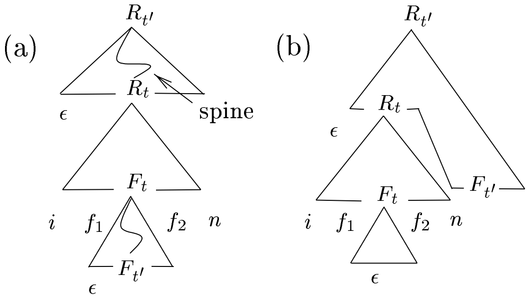


Figure 5.2: Wrapping of auxiliary trees when computing the prefix probability

probabilities of items associated with equal or smaller portions of the input.

In specifying the equations, we exploit techniques used in the parsing of incomplete input [Lang, 1988]. This allows us to compute the prefix probability as a by-product of computing the inside probability.

In order to avoid the problem of nontermination outlined above, we transform our equations to remove infinite recursion, while preserving the correctness of the probability computation. The transformation of the equations is explained as follows. For an item I, the **span** of I, written $\mathcal{S}(I)$, is the 4-tuple representing the 4 input positions in I. We will define an equivalence relation on spans that relates to the portion of the input that is covered. The transformations that we apply to our equations produce two new sets of equations. The first set of equations are concerned with all possible decompositions of a given item I into set of items of which one has a span equivalent to that of I and the others have an empty span. Equations in this set represent endless recursion. The system of all such equations can be solved independently of the actual input w. This is done once for a given grammar.

The second set of equations have the property that, when evaluated, recursion always terminates. The evaluation of these equations computes the probability of the input string modulo the computation of some parts of the derivation that do not contribute to the input itself. Combination of the second set of equations with the solutions obtained from the first set allows the effective computation of the prefix probability.

5.4 Computing Prefix Probabilities

This section develops an algorithm for the computation of prefix probabilities for probabilistic TAGs.

5.4.1 General equations

The prefix probability is given by:

$$\sum_{w \in \Sigma^*} \Pr(a_1 \cdots a_n w) \quad = \quad \sum_{t \in \mathcal{I}} P([t, 0, n, -, -]),$$

where P is a function over items recursively defined as follows:

$$P([t,i,j,f_1,f_2]) \ = P([R_t,i,j,f_1,f_2]); \tag{5.1}$$

$$\begin{aligned} &P([\alpha N,i,j,-,-]) \ = \\ &\quad \sum_{k(i \leq k \leq j)} P([\alpha,i,k,-,-]) \cdot P([N,k,j,-,-]), \\ &\quad \text{if } \alpha \neq \epsilon \wedge \neg \mathit{dft}(\alpha N); \end{aligned} \tag{5.2}$$

$$\begin{aligned} &P([\alpha N,i,j,f_1,f_2]) \ = \\ &\quad \sum_{k(i \leq k \leq f_1)} P([\alpha,i,k,-,-]) \cdot P([N,k,j,f_1,f_2]), \\ &\quad \text{if } \alpha \neq \epsilon \wedge \mathit{dft}(N); \end{aligned} \tag{5.3}$$

$$\begin{aligned} &P([\alpha N,i,j,f_1,f_2]) \ = \\ &\quad \sum_{k(f_2 \leq k \leq j)} P([\alpha,i,k,f_1,f_2]) \cdot P([N,k,j,-,-]), \\ &\quad \text{if } \alpha \neq \epsilon \wedge \mathit{dft}(\alpha); \end{aligned} \tag{5.4}$$

$$\begin{aligned} &P([N,i,j,f_1,f_2]) \ = \\ &\quad \phi(\mathbf{nil},N) \cdot P([\mathit{cdn}(N),i,j,f_1,f_2]) \ + \\ &\quad \sum_{f'_1,f'_2(i \leq f'_1 \leq f_1 \wedge f_2 \leq f'_2 \leq j)} P([\mathit{cdn}(N),f'_1,f'_2,f_1,f_2]) \ \cdot \\ &\qquad\qquad \sum_{t \in \mathcal{A}} \phi(t,N) \cdot P([t,i,j,f'_1,f'_2]), \\ &\quad \text{if } N \in V \wedge \mathit{dft}(N); \end{aligned} \tag{5.5}$$

$$\begin{aligned} &P([N,i,j,-,-]) \ = \\ &\quad \phi(\mathbf{nil},N) \cdot P([\mathit{cdn}(N),i,j,-,-]) \ + \\ &\quad \sum_{f'_1,f'_2(i \leq f'_1 \leq f'_2 \leq j)} P([\mathit{cdn}(N),f'_1,f'_2,-,-]) \ \cdot \\ &\qquad\qquad \sum_{t \in \mathcal{A}} \phi(t,N) \cdot P([t,i,j,f'_1,f'_2]), \\ &\quad \text{if } N \in V \wedge \neg \mathit{dft}(N); \end{aligned} \tag{5.6}$$

$$\begin{aligned} &P([a,i,j,-,-]) \ = \\ &\quad \delta(i+1=j \wedge a_j = a) + \delta(i=j=n); \end{aligned} \tag{5.7}$$

$$P([\bot,i,j,f_1,f_2]) \ = \delta(i=f_1 \wedge j=f_2); \tag{5.8}$$

$$P([\epsilon,i,j,-,-]) \ = \delta(i=j). \tag{5.9}$$

Term $P([t,i,j,f_1,f_2])$ gives the inside probability of all possible trees derived from elementary tree t, having the indicated span over the input. This is decomposed into the contribution of each single node of t in equations (5.1) through (5.6). In equations (5.5) and (5.6) the contribution of a node N is determined by the combination of the inside probabilities of N's children and by all possible adjunctions at N. In (5.7) we recognize some terminal symbol if it occurs in the prefix, or ignore its contribution to the span if it occurs after the last symbol of the prefix. Crucially, this step allows us to reduce the computation of prefix probabilities to the computation of inside probabilities.

5.4.2 Terminating equations

In general, the recursive equations (5.1) to (5.9) are not directly computable. This is because the value of $P([A,i,j,f,f'])$ might indirectly depend on itself, giving rise to non-termination. We therefore rewrite the equations.

We define an equivalence relation over spans, that expresses when two items are associated with equivalent portions of the input:

$$
\begin{aligned}
&(i',j',f'_1,f'_2) \approx (i,j,f_1,f_2) \text{ if and only if}\\
&\quad ((i',j') = (i,j)) \wedge\\
&\qquad ((f'_1,f'_2) = (f_1,f_2) \vee\\
&\qquad ((f'_1 = f'_2 = i \vee f'_1 = f'_2 = j \vee f'_1 = f'_2 = -) \wedge\\
&\qquad\ (f_1 = f_2 = i \vee f_1 = f_2 = j \vee f_1 = f_2 = -)))
\end{aligned}
$$

We introduce two new functions P_{low} and P_{split}. When evaluated on some item I, P_{low} recursively calls itself as long as some other item I' with a given elementary tree as its first component can be reached, such that $\mathcal{S}(I) \approx \mathcal{S}(I')$. P_{low} returns 0 if the actual branch of recursion cannot eventually reach such an item I', thus removing the contribution to the prefix probability of that branch. If item I' is reached, then P_{low} switches to P_{split}. Complementary to P_{low}, function P_{split} tries to decompose an argument item I into items I' such that $\mathcal{S}(I) \not\approx \mathcal{S}(I')$. If this is not possible through the actual branch of recursion, P_{split} returns 0. If decomposition is indeed possible, then we start again with P_{low} at items produced by the decomposition. The effect of this intermixing of function calls is the simulation of the original function P, with P_{low} being called only on potentially nonterminating parts of the computation, and P_{split} being called on parts that are guaranteed to terminate.

Consider some derivation tree spanning some portion of the input string, and the associated derivation tree τ. There must be a unique elementary tree which is represented by a node in τ that is the "lowest" one that entirely spans the portion of the input of interest. (This node might be the root of τ itself.) Then, for each $t \in \mathcal{A}$ and for each i, j, f_1, f_2 such that $i < j$ and $i \leq f_1 \leq f_2 \leq j$, we must have:

$$P([t, i, j, f_1, f_2]) \;=\; \sum_{t' \,\in\, \mathcal{A},\, f_1', f_2'((i,j,f_1',f_2') \,\approx\, (i,j,f_1,f_2))} P_{low}([t, i, j, f_1, f_2],\ [t', f_1', f_2']). \tag{5.10}$$

Similarly, for each $t \in \mathcal{I}$ and for each i, j such that $i < j$, we must have:

$$P([t, i, j, -, -]) \;=\; \sum_{t' \,\in\, \{t\} \cup \mathcal{A},\, f \,\in\, \{-, i, j\}} P_{low}([t, i, j, -, -],\ [t', f, f]). \tag{5.11}$$

The reason why P_{low} keeps a record of indices f_1' and f_2', i.e., the spanning of the foot node of the lowest tree (in the above sense) on which P_{low} is called, will become clear later, when we introduce equations (5.29) and (5.30).

We define $P_{low}([t, i, j, f_1, f_2], [t', f_1', f_2'])$ and $P_{low}([\alpha, i, j, f_1, f_2], [t', f_1', f_2'])$ for $i < j$ and $(i, j, f_1, f_2) \approx (i, j, f_1', f_2')$, as follows. $P_{low}([t,\ i,\ j,\ f_1,\ f_2],\ [t', f_1', f_2'])\ =$ (12)

$$\begin{aligned}
&P_{low}([R_t,\ i,\ j,\ f_1,\ f_2],\ [t', f_1', f_2']) \ + \\
&\delta((t, f_1, f_2) = (t', f_1', f_2')) \ \cdot \\
&\qquad P_{split}([R_t,\ i,\ j,\ f_1,\ f_2]);
\end{aligned}$$

$$\begin{aligned}
P_{low}([\alpha N, i, j, -, -],\ [t, f_1', f_2']) \ &= \qquad (13) \\
&P_{low}([\alpha, i, j, -, -],\ [t, f_1', f_2']) \ \cdot \\
&\qquad P([N, j, j, -, -]) \ + \\
&P([\alpha, i, i, -, -]) \ \cdot
\end{aligned}$$

$$\begin{aligned}
&\qquad P_{low}([N, i, j, -, -],\ [t, f_1', f_2']),\\
&\text{if } \alpha \neq \epsilon \wedge \neg \mathit{dft}(\alpha N);
\end{aligned}$$

$$\begin{aligned}
&P_{low}([\alpha N, i, j, f_1, f_2],\ [t, f_1', f_2'])\ =\\
&\quad \delta(f_1 = j) \cdot P_{low}([\alpha, i, j, -, -],\ [t, f_1', f_2'])\ \cdot\\
&\qquad\quad P([N, j, j, f_1, f_2])\ +\\
&\quad P([\alpha, i, i, -, -])\ \cdot\\
&\qquad\quad P_{low}([N, i, j, f_1, f_2],\ [t, f_1', f_2']),\\
&\quad \text{if } \alpha \neq \epsilon \wedge \mathit{dft}(N);
\end{aligned} \tag{14}$$

$$\begin{aligned}
&P_{low}([\alpha N, i, j, f_1, f_2],\ [t, f_1', f_2'])\ =\\
&\quad P_{low}([\alpha, i, j, f_1, f_2],\ [t, f_1', f_2'])\ \cdot\\
&\qquad\quad P([N, j, j, -, -])\ +\\
&\quad \delta(i = f_2) \cdot P([\alpha, i, i, f_1, f_2])\ \cdot\\
&\qquad\quad P_{low}([N, i, j, -, -],\ [t, f_1', f_2']),\\
&\quad \text{if } \alpha \neq \epsilon \wedge \mathit{dft}(\alpha);
\end{aligned} \tag{15}$$

$$\begin{aligned}
&P_{low}([N, i, j, f_1, f_2],\ [t, f_1', f_2'])\ =\\
&\quad \phi(\mathbf{nil}, N)\ \cdot\\
&\qquad\quad P_{low}([\mathit{cdn}(N), i, j, f_1, f_2],\ [t, f_1', f_2'])\ +\\
&\quad P_{low}([\mathit{cdn}(N), i, j, f_1, f_2],\ [t, f_1', f_2'])\ \cdot\\
&\qquad\quad \textstyle\sum_{t' \in \mathcal{A}} \phi(t', N) \cdot P([t', i, j, i, j])\ +\\
&\quad P([\mathit{cdn}(N), f_1, f_2, f_1, f_2])\ \cdot\\
&\qquad \sum_{t' \in \mathcal{A}} \phi(t', N) \cdot P_{low}([t', i, j, f_1, f_2],\ [t, f_1', f_2']),\\
&\quad \text{if } N \in V \wedge \mathit{dft}(N);
\end{aligned} \tag{16}$$

$$\begin{aligned}
&P_{low}([N, i, j, -, -],\ [t, f_1', f_2'])\ =\\
&\quad \phi(\mathbf{nil}, N)\ \cdot\\
&\qquad\quad P_{low}([\mathit{cdn}(N), i, j, -, -],\ [t, f_1', f_2'])\ +\\
&\quad P_{low}([\mathit{cdn}(N), i, j, -, -],\ [t, f_1', f_2'])\ \cdot
\end{aligned} \tag{17}$$

$$
\begin{aligned}
&\textstyle\sum_{t' \in \mathcal{A}} \phi(t', N) \cdot P([t', i, j, i, j]) \ + \\
&\sum_{f_1'', f_2'' (f_1'' = f_2'' = i \vee f_1'' = f_2'' = j)} P([cdn(N), f_1'', f_2'', -, -]) \ \cdot \\
&\quad \sum_{t' \in \mathcal{A}} \phi(t', N) \cdot P_{low}([t', i, j, f_1'', f_2''], \ [t, f_1', f_2']), \\
&\text{if } N \in V \wedge \neg dft(N);
\end{aligned}
$$

$$P_{low}([a, i, j, -, -], \ [t, f_1', f_2']) = 0; \tag{18}$$

$$P_{low}([\bot, i, j, f_1, f_2], \ [t, f_1', f_2']) = 0; \tag{19}$$

$$P_{low}([\epsilon, i, j, -, -], \ [t, f_1', f_2']) = 0. \tag{20}$$

The definition of P_{low} parallels the one of P given in §5.4.1. In (5.12), the second term in the right-hand side accounts for the case in which the tree we are visiting is the “lowest” one on which P_{low} should be called. Note how in the above equations P_{low} must be called also on nodes that do not dominate the footnode of the elementary tree they belong to (cf. the definition of $\approx$). Since no call to P_{split} is possible through the terms in (5.18), (5.19) and (5.20), we must set the right-hand side of these equations to 0.

The specification of $P_{split}([\alpha, i, j, f_1, f_2])$ is given below. Again, the definition parallels the one of P given in §5.4.1.

$$
\begin{aligned}
&P_{split}([\alpha N, i, j, -, -]) \ = \\
&\quad \sum_{k (i < k < j)} P([\alpha, i, k, -, -]) \cdot P([N, k, j, -, -]) \ + \\
&\quad P_{split}([\alpha, i, j, -, -]) \cdot P([N, j, j, -, -]) \ + \\
&\quad P([\alpha, i, i, -, -]) \cdot P_{split}([N, i, j, -, -]), \\
&\quad \text{if } \alpha \neq \epsilon \wedge \neg dft(\alpha N);
\end{aligned} \tag{5.21}
$$

$$
\begin{aligned}
&P_{split}([\alpha N, i, j, f_1, f_2]) \ = \\
&\quad \sum_{k (i < k \leq f_1 \wedge k < j)} P([\alpha, i, k, -, -]) \cdot P([N, k, j, f_1, f_2]) \ + \\
&\quad \delta(f_1 = j) \cdot P_{split}([\alpha, i, j, -, -]) \ \cdot \\
&\qquad\qquad P([N, j, j, f_1, f_2]) \ + \\
&\quad P([\alpha, i, i, -, -]) \cdot P_{split}([N, i, j, f_1, f_2]), \\
&\quad \text{if } \alpha \neq \epsilon \wedge dft(N);
\end{aligned} \tag{5.22}
$$

$$
\begin{aligned}
& P_{split}([\alpha N, i, j, f_1, f_2]) \; = \\
& \quad \sum_{k (i < k \wedge f_2 \leq k < j)} P([\alpha, i, k, f_1, f_2]) \cdot P([N, k, j, -, -]) \; + \\
& \quad P_{split}([\alpha, i, j, f_1, f_2]) \cdot P([N, j, j, -, -]) \; + \\
& \quad \delta(i = f_2) \cdot P([\alpha, i, i, f_1, f_2]) \cdot \\
& \qquad P_{split}([N, i, j, -, -]), \\
& \quad \text{if } \alpha \neq \epsilon \wedge \mathit{dft}(\alpha);
\end{aligned} \tag{5.23}
$$

$$
\begin{aligned}
& P_{split}([N, i, j, f_1, f_2]) \; = \\
& \quad \phi(\mathbf{nil}, N) \cdot P_{split}([\mathit{cdn}(N), i, j, f_1, f_2]) \; + \\
& \quad \sum_{\substack{f'_1, f'_2 \; (i \leq f'_1 \leq f_1 \wedge f_2 \leq f'_2 \leq j \; \wedge \\ (f'_1, f'_2) \neq (i, j) \wedge (f'_1, f'_2) \neq (f_1, f_2))}} P([\mathit{cdn}(N), f'_1, f'_2, f_1, f_2]) \; \cdot \\
& \qquad \sum_{t \in \mathcal{A}} \phi(t, N) \cdot P([t, i, j, f'_1, f'_2]) \; + \\
& \quad P_{split}([\mathit{cdn}(N), i, j, f_1, f_2]) \; \cdot \\
& \qquad \sum_{t \in \mathcal{A}} \phi(t, N) \cdot P([t, i, j, i, j]), \\
& \quad \text{if } N \in V \wedge \mathit{dft}(N);
\end{aligned} \tag{5.24}
$$

$$
\begin{aligned}
& P_{split}([N, i, j, -, -]) \; = \\
& \quad \phi(\mathbf{nil}, N) \cdot P_{split}([\mathit{cdn}(N), i, j, -, -]) \; + \\
& \quad \sum_{\substack{f'_1, f'_2 \; (i \leq f'_1 \leq f'_2 \leq j \wedge (f'_1, f'_2) \neq (i, j) \; \wedge \\ \neg (f'_1 = f'_2 = i \vee f'_1 = f'_2 = j))}} P([\mathit{cdn}(N), f'_1, f'_2, -, -]) \; \cdot \\
& \qquad \sum_{t \in \mathcal{A}} \phi(t, N) \cdot P([t, i, j, f'_1, f'_2]) \; + \\
& \quad P_{split}([\mathit{cdn}(N), i, j, -, -]) \; \cdot \\
& \qquad \sum_{t \in \mathcal{A}} \phi(t, N) \cdot P([t, i, j, i, j]), \\
& \quad \text{if } N \in V \wedge \neg \mathit{dft}(N);
\end{aligned} \tag{5.25}
$$

$$
P_{split}([a, i, j, -, -]) = \delta(i + 1 = j \wedge a_j = a); \tag{5.26}
$$

$$
P_{split}([\perp, i, j, f_1, f_2]) = 0; \tag{5.27}
$$

$$
P_{split}([\epsilon, i, j, -, -]) = 0. \tag{5.28}
$$

We can now separate those branches of recursion that terminate on the given input from the cases of endless recursion. We assume below that $P_{split}([R_t, i, j, f'_1, f'_2]) > 0$. Even if this is not always valid, for the purpose of deriving the equations below, this assumption does not lead to invalid results. We define a new function P_{outer}, which accounts for probabilities of subderivations that do not derive any words in the prefix, but contribute structurally to its derivation:

$$\begin{aligned} P_{outer}([t, i, j, f_1, f_2],\ [t', f'_1, f'_2]) \ &= \\ &\frac{P_{low}([t, i, j, f_1, f_2],\ [t', f'_1, f'_2])}{P_{split}([R_{t'}, i, j, f'_1, f'_2])}; \end{aligned} \tag{5.29}$$

$$\begin{aligned} P_{outer}([\alpha, i, j, f_1, f_2],\ [t', f'_1, f'_2]) \ &= \\ &\frac{P_{low}([\alpha, i, j, f_1, f_2],\ [t', f'_1, f'_2])}{P_{split}([R_{t'}, i, j, f'_1, f'_2])}. \end{aligned} \tag{5.30}$$

We can now eliminate the infinite recursion that arises in (5.10) and (5.11) by rewriting $P([t, i, j, f_1, f_2])$ in terms of P_{outer}:

$$\begin{aligned} P([t, i, j, f_1, f_2]) \ &= \\ &\sum_{t' \in \mathcal{A},\, f'_1, f'_2 ((i,j,f'_1,f'_2) \approx (i,j,f_1,f_2))} P_{outer}([t, i, j, f_1, f_2],\ [t', f'_1, f'_2]) \cdot \\ &\qquad P_{split}([R_{t'}, i, j, f'_1, f'_2]); \end{aligned} \tag{5.31}$$

$$\begin{aligned} P([t, i, j, -, -]) \ &= \\ &\sum_{t' \in \{t\} \cup \mathcal{A},\, f \in \{-, i, j\}} P_{outer}([t, i, j, -, -],\ [t', f, f]) \cdot \\ &\qquad P_{split}([R_{t'}, i, j, f, f]). \end{aligned} \tag{5.32}$$

Equations for P_{outer} will be derived in the next subsection.

In summary, terminating computation of prefix probabilities should be based on equations (5.31) and (5.32), which replace (5.1), along with equations (5.2) to (5.9) and all the equations for P_{split}.

5.4.3 Off-line Equations

In this section we derive equations for function P_{outer} introduced in §5.4.2 and deal with all remaining cases of equations that cause infinite recursion.

In some cases, function P can be computed independently of the actual input. For any $i < n$ we can consistently define the following quantities, where $t \in \mathcal{I} \cup \mathcal{A}$ and $\alpha \in V^{\perp}$ or $cdn(N) = \alpha\beta$ for some N and β:

$$\begin{aligned} H_t &= P([t, i, i, f, f]); \\ H_\alpha &= P([\alpha, i, i, f', f']), \end{aligned}$$

where $f = i$ if $t \in \mathcal{A}$, $f = -$ otherwise, and $f' = i$ if $dft(\alpha)$, $f = -$ otherwise. Thus, H_t is the probability of all derived trees obtained from t, with no lexical node at their yields. Quantities H_t and H_α can be computed by means of a system of equations which can be directly obtained from equations (5.1) to (5.9). Similar quantities as above must be introduced for the case $i = n$. For instance, we can set $H'_t = P([t, n, n, f, f])$, f specified as above, which gives the probability of all derived trees obtained from t (with no restriction at their yields).

Function P_{outer} is also independent of the actual input. Let us focus here on the case $f_1, f_2 \not\in \{i, j, -\}$ (this enforces $(f_1, f_2) = (f'_1, f'_2)$ below). For any $i, j, f_1, f_2 < n$, we can consistently define the following quantities.

$$\begin{aligned} L_{t,t'} &= P_{outer}([t, i, j, f_1, f_2],\ [t', f'_1, f'_2]); \\ L_{\alpha,t'} &= P_{outer}([\alpha, i, j, f_1, f_2],\ [t', f'_1, f'_2]). \end{aligned}$$

In the case at hand, $L_{t,t'}$ is the probability of all derived trees obtained from t such that (i) no lexical node is found at their yields; and (ii) at some 'unfinished' node dominating the foot of t, the probability of the adjunction of t' has already been accounted for, but t' itself has not been adjoined.

It is straightforward to establish a system of equations for the computation of $L_{t,t'}$ and $L_{\alpha,t'}$, by rewriting equations (5.12) to (5.20) according to (5.29) and (5.30). For instance, combining (5.12) and (5.29) gives (using the above assumptions on f_1 and f_2):

$$L_{t,t'} = L_{R_t,t'} + \delta(t = t').$$

Also, if $\alpha \neq \epsilon$ and $dft(N)$, combining (5.14) and (5.30) gives (again, using previous assumptions on f_1 and f_2; note that the H_α's are known terms here):

$$L_{\alpha N,t'} = H_\alpha \cdot L_{N,t'}.$$

For any $i, f_1, f_2 < n$ and $j = n$, we also need to define:

$$\begin{aligned} L'_{t,t'} &= P_{outer}([t, i, n, f_1, f_2],\ [t', f'_1, f'_2]); \\ L'_{\alpha,t'} &= P_{outer}([\alpha, i, n, f_1, f_2],\ [t', f'_1, f'_2]). \end{aligned}$$

Here $L'_{t,t'}$ is the probability of all derived trees obtained from t with a node dominating the foot node of t, that is an adjunction site for t' and is 'unfinished' in the same sense as above, and with lexical nodes only in the portion of the tree to the right of that node. When we drop our assumption on f_1 and f_2, we must (pre)compute in addition terms of the form $P_{outer}([t, i, j, i, i],\ [t', i, i])$ and $P_{outer}([t, i, j, i, i],\ [t', j, j])$ for $i < j < n$, $P_{outer}([t, i, n, f_1, n],$ $[t', f'_1, f'_2])$ for $i < f_1 < n$, $P_{outer}([t, i, n, n, n],\ [t', f'_1, f'_2])$ for $i < n$, and similar. Again, these are independent of the choice of i, j and f_1. Full treatment is omitted due to length restrictions.

5.5 Remarks on Complexity

Function P_{split} is the core of the method. Its equations can be directly translated into an effective algorithm, using standard functional memoization or other tabular techniques. It is easy to see that such an algorithm can be made to run in time $\mathcal{O}(n^6)$, where n is the length of the input prefix.

All the quantities introduced in §5.4.3 (H_t, $L_{t,t'}$, etc.) are independent of the input and should be computed off-line, using the system of equations that can be derived as indicated. For quantities H_t we have a non-linear system, since equations (2) to (6) contain quadratic terms. Solutions can then be approximated to any degree of precision using standard iterative methods, as for instance those exploited in [Stolcke, 1995]. Under the hypothesis that the grammar is consistent, that is $\Pr(L(G)) = 1$, all quantities H'_t and H'_α evaluate to one. For quantities $L_{t,t'}$ and the like, §5.4.3 provides linear systems whose solutions can easily be obtained using standard methods. Note also that quantities $L_{\alpha,t'}$ are only used in the off-line computation of quantities $L_{t,t'}$, they do not need to be stored for the computation of prefix probabilities (compare equations for $L_{t,t'}$ with (5.31) and (5.32)).

We can easily develop implementations of our method that can compute prefix probabilities incrementally. That is, after we have computed the prefix probability for a prefix

$a_1 \cdots a_n$, on input a_{n+1} we can extend the calculation to prefix $a_1 \cdots a_n a_{n+1}$ without having to recompute all intermediate steps that do not depend on a_{n+1}. This step takes time $\mathcal{O}(n^5)$.

In this chapter we have assumed that the parameters of the probabilistic TAG have been previously estimated. In practice, smoothing to avoid sparse data problems plays an important role. Smoothing can be handled for prefix probability computation in the following ways. Discounting methods for smoothing simply produce a modified STAG model which is then treated as input to the prefix probability computation. Smoothing using methods such as deleted interpolation which combine class-based models with word-based models to avoid sparse data problems have to be handled by a cognate interpolation of prefix probability models.

5.6 Computing the Off-line Probabilities

To compute the off-line probabilities we simply use the results already shown in Chapter 2. The stochastic expectation matrix computed for the input probabilistic TAG can be used directly to infer the probability of the 'jump' between two nodes in separate elementary trees. The probabilities can be directly examined by computing the expectation matrix assuming that each tree in the grammar can be the root of a (sub)derivation and then examining the eigenvalues corresponding to each node that accepts adjunction that can be reached from the node in that tree.

We will include more detailed equations for this computation in a later version of this document.

5.7 Conclusion

The result here is presented in a theoretical framework and to address implementational issues we give precise steps for the computation of prefix probabilities during the steps taken by a parser in Appendix B.

The main result in this chapter is an algorithm to compute such prefix probabilities given a probabilistic Tree Adjoining Grammar (TAG). The algorithm achieves the required computation in $\mathcal{O}(n^6)$ time. The probability of subderivations that do not derive any words in the prefix, but contribute structurally to its derivation, are precomputed to achieve termination. This algorithm enables existing corpus-based estimation techniques for probabilistic TAGs to be used for language modeling. We show how to parse substrings in the input and the off-line computation of probabilities for portions of the derivation that are indirectly used but which do not directly contribute any lexical items towards the parse of the initial substring.

Chapter 6

Proposed Work

The goal in this dissertation have been two-fold: to develop the theory of probabilistic Tree Adjoining Grammars (TAGs) and to present some practical results in the form of efficient parsing and estimation algorithms for statistical parsing.

The overall goal of developing the theory of probabilistic TAGs is to provide a simple, mathematically and linguistically well-formed probabilistic framework for topics in statistical parsing.

The practical results in parsing and estimation of probabilistic TAGs were developed with a view towards an increasingly unsupervised approach to the training of statistical parsers and language models.

We have shown in previous chapters the following results:

- An algorithm for determining deficiency in a generative model for probabilistic TAGs (Chapter 2).
- A chart based head-corner parsing algorithm for probabilistic TAGs. (Chapter 3)
- A probability model for statistical parsing and a co-training method for training this parser which combines labeled and unlabeled data. (Chapter 4)
- An algorithm for computing prefix probabilities which can be used to predict the word most likely to occur after an initial substring of the input. (Chapter 5)

In this chapter we lay out the work that is a logical extension of the work presented including some addition work to be done in the evaluation of some of the methods proposed in this work.

Much of the supporting work that pertains to the proposed work in this chapter has been put in the appendices. We refer to these appendices when necessary in the following text.

To summarize, the proposed work consists of the following items:

- More experiments for the co-training algorithm, and experiments comparing the prefix probability parser with a trigram model.

- Using improved methods of classification of verbs to create dictionaries that can be directly used in the co-training algorithm.

- To combine ideas from initial substring parsing and the combination of multiple classifiers to form a model for parsing which incrementally reduces the amount of labeled training data that is required for building an effective parser.

6.1 Further Evaluation of Co-training

The co-training algorithm presented in Chapter 4 which combined labeled and unlabeled data was evaluated on a particular set of data which was selected with the view of minimizing the amount of labeled data. We propose to do a separate evaluation of the co-training algorithm on a larger set of labeled and unlabeled data. In particular the 1M word WSJ Penn Treebank corpus as the labeled data and a 23M word WSJ corpus as the unlabeled training set. As mentioned in the chapter about the co-training algorithm, the particular method proposed allows one to experiment with larger unlabeled data sets as training material. In particular, the use of a small cache which is the only part which is iteratively parsed in each step and also the fact that the rest of the training data is processed only once in the procedure ensures that larger data sets (in this case, 23M words) can be effectively used by the proposed co-training method.

6.2 Evaluation of the Prefix Probability Parser

Evaluation of the prefix probability algorithm by comparing it with a trigram language model. In Appendix B we describe the changes to be made in conventional TAG parsing algorithms to parse initial substrings and compute their probabilities. We plan to modify the algorithm given in Chapter 3 in the fashion described in Appendix B. This will also unify the slightly different notation used in Chapters 3 and 5. The evaluation will consist of the usual comparision of perplexity values between the model proposed and the perplexity values obtained from backed-off trigram models on the same data. Note that the combination of labeled and unlabeled data for training the probability model will allow us to effectively compare our model with trigram models in contrast with previous efforts [Chelba and Jelinek, 1998] which limit the size of the trigram models to the size of the training data available. We would like to note that it would be a more effective test to compare word error rates rather than perplexity, but this experiment needs access to data sets which are not freely available.

6.3 Learning TAG Lexicons from Corpora

In Chapter 4 we describe a method of learning a parser by combining labeled and unlabeled data. One prerequisite of the method was the use of a dictionary of possible trees for each word in the training data. In the experiments we have done, this dictionary was collected directly from the labeled data and no further learning was done on the unlabeled set. It is easy to see that further learning algorithms can be useful in inducing such a dictionary. This would help in increasing the amount of lexical information accessed by the component of the learning algorithm that bootstraps from the unlabeled data.

In order to improve the dictionaries used, we need better methods for grouping words into classes. This is usually handled by clustering algorithms which look at the immediate context of the words to group them into classes. We use an alternative approach to the classification of words that are predicates (usually the more informative words in a TAG dictionary). We classify verbs by their argument structure thus effectively sharing dictionary entries learnt from labeled data for words with the same argument structure.

We present two techniques in Appendix C that help with this classification. Both of these techniques learn from counts collected from all occurences of the word in the corpus. One technique learns subcategorization information while the other learns to group together verbs that might have the same subcategorization but which belong to different classes.

In proposed work, we hope to include this information into the construction of dictionaries in the co-training algorithm given in Chapter 4.

6.4 Unsupervised Learning of Parsers

In our work we have concentrated on learning from a combination of labeled and unlabeled data. The classification approach of simultaneously learning the assignment of structure to a word while correctly finding attachments (see Chapter 4) can be generalized to a left to right model as described in the prefix probability chapter (Chapter 5). A generalization of the co-training approach which includes more than two classifiers can be used to obtain a general form of the skip-and-attach model described in [Chelba and Jelinek, 1998] and in [Pietra et al., 1994]. Such a model can be defined as a trigram LTAG model which computes attachments based on several classifiers combining those defined in Chapter 4 and Appendix C. This is a more experimental proposal and although we have worked out some of the mathematical details behind the model, some of the more practical aspects of its implementation have still to be worked out in greater detail.

Appendix A

Factors Affecting Parsing Efficiency in TAG Parsing

In this section we report on some practical experiments where we parse 2250 sentences from the Wall Street Journal using this parser. In these experiments the parser is run without any statistical pruning; it produces all valid parses for each sentence in the form of a shared derivation forest. The parser uses a large Treebank Grammar with 6789 tree templates with about $120,000$ lexicalized trees. The results suggest that the observed complexity of parsing for LTAG is dominated by factors other than sentence length.

The particular experiments that we report on in this chapter were chosen to discover certain facts about LTAG parsing in a practical setting. Specifically, we wanted to discover the importance of the worst-case results for LTAG parsing in practice. Let us take the parsing algorithm introduced in this chapter: the parsing time complexity of this algorithm for various types of grammars are as follows (for input of length n):

$O(n^6)$ - TAGs for inherently ambiguous languages

$O(n^4)$ - unambiguous TAGs

$O(n)$ - bounded state TAGs e.g. the usual grammar G where $L(G) = \{a^n\, b^n\, e\, c^n\, d^n \mid n \geq 0\}$ (see [Joshi et al., 1975])

The grammar factors are as follows: the parser takes $O(|A||I \cup A|Nn^6)$ worst case time and

$O(|A \cup I| N n^4)$ worst case space, where n is the length of the input, A is the set of auxiliary trees, I is the set of initial trees and N is maximum number of nodes in an elementary tree.

Given these worst case estimates we wish to explore what the observed times might be for a TAG parser. It is not our goal here to compare different TAG parsing algorithms, rather it is to discover what kinds of factors can contribute to parsing time complexity. Of course, a natural-language grammar that is large and complex enough to be used for parsing real-world text is typically neither unambiguous nor bounded in state size. It is important to note that in this chapter we are not concerned with *parsing accuracy*, rather we want to explore *parsing efficiency*. This is why we do not pursue any pruning while parsing using statistical methods. Instead we produce a shared derivation forest for each sentence which stores, in compact form, all derivations for each sentence. This helps us evaluate our TAG parser for time and space efficiency. The experiments reported here are also useful for statistical parsing using TAG since discovering the source of grammar complexity in parsing can help in finding the right *figures-of-merit* for effective pruning in a statistical parser.

A.0.1 LTAG **Treebank Grammar**

The grammar we used for our experiments was a LTAG Treebank Grammar which was automatically extracted from Sections 02–21 of the Wall Street Journal Penn Treebank II corpus [Marcus et al., 1993]. The extraction tool [Xia, 1999] converted the *derived* trees of the Treebank into *derivation* trees in LTAG which represent the attachments of lexicalized elementary trees. There are 6789 tree templates in the grammar with $47,752$ tree nodes. Each word in the corpus selects some set of tree templates. The total number of lexicalized trees is $123,039$. The total number of word types in the lexicon is $44,215$. The average number of trees per word type is 2.78. However, this average is misleading since it does not consider the frequency with which words that select a large number of trees occur in the corpus. In Figure A.1 we see that many frequently seen words can select a large number of trees. Finally, some lexicalized trees from the grammar are shown in Figure A.2.

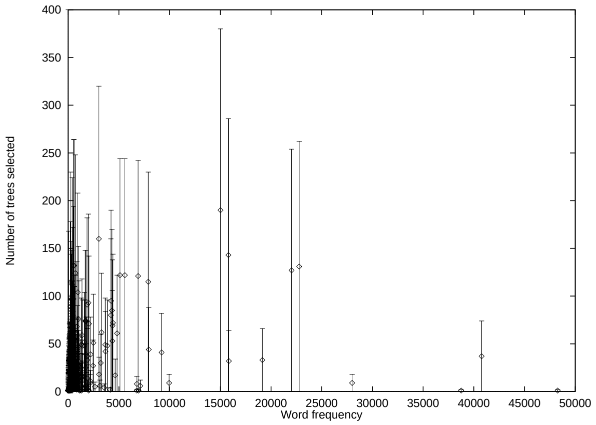
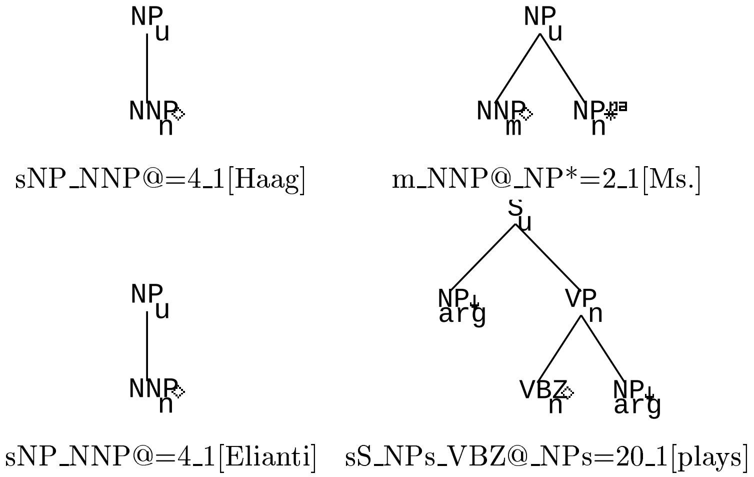


Figure A.1: Number of trees selected plotted against words with a particular frequency. (x-axis: words of frequency x; y-axis: number of trees selected, error bars indicate least and most ambiguous word of a particular frequency x)



sNP_NNP@=4_1[Haag] m_NNP@_NP*=2_1[Ms.]

sNP_NNP@=4_1[Elianti] sS_NPs_VBZ@_NPs=20_1[plays]

Figure A.2: Example lexicalized elementary trees from the Treebank Grammar. They are shown in the usual notation: $\diamond$ = *anchor*, $\downarrow$= *substitution node*, $*$ = *footnode*, **na** = *null-adjunction constraint*. These trees can be combined using substitution and adjunction to parse the sentence *Ms. Haag plays Elianti.*

A.0.2 Syntactic Lexical Ambiguity

In a fully lexicalized grammar such as LTAG the combinations of trees (by substitution and adjunction) can be thought of as *attachments*. It is this perspective that allows us to define the parsing problem in two steps [Joshi and Schabes, 1991]:

1. Assigning a set of lexicalized structures to each word in the input sentence.
2. Finding the correct attachments between these structures to get all parses for the sentence.

In this section we will try to find which of these factors determines parsing complexity when finding all parses in an LTAG parser.

To test the performance of LTAG parsing on a realistic corpus using a large grammar (described above) we parsed 2250 sentences from the Wall Street Journal using the lexicalized grammar described in Section A.0.1. All of these sentences were of length 21 words or less. These sentences were taken from the same sections (02-21) of the Treebank from which the original grammar was extracted. This was done to avoid the complication of using default rules for unknown words.

In all of the experiments reported here, the parser produces all parses for each sentence. It produces a shared derivation forest for each sentence which stores, in compact form, all derivations for each sentence.

We found that the observed complexity of parsing for LTAG is dominated by factors other than sentence length.[1] Figure A.3 shows the time taken in seconds by the parser plotted against sentence length. We see a great deal of variation in timing for the same sentence length, especially for longer sentences.

We wanted to find the relevant variable other than sentence length which would be the right predictor of parsing time complexity. There can be a large variation in syntactic lexical ambiguity which might be a relevant factor in parsing time complexity. To draw this out, in Figure A.4 we plotted the number of trees selected by a sentence against

[1] Note that the precise number of edges proposed by the parser and other common indicators of complexity can be obtained only while or after parsing. We are interested in *predicting* parsing complexity.

the time taken to parse that sentence. By examining this graph we can visually infer that the number of trees selected is a better predictor of increase in parsing complexity than sentence length. We can also compare numerically the two hypotheses by computing the coefficient of determination (R^2) for the two graphs. We get a R^2 value of 0.65 for Figure A.3 and a value of 0.82 for Figure A.4. Thus, we infer that it is the syntactic lexical ambiguity of the words in the sentence which is the major contributor to parsing time complexity.

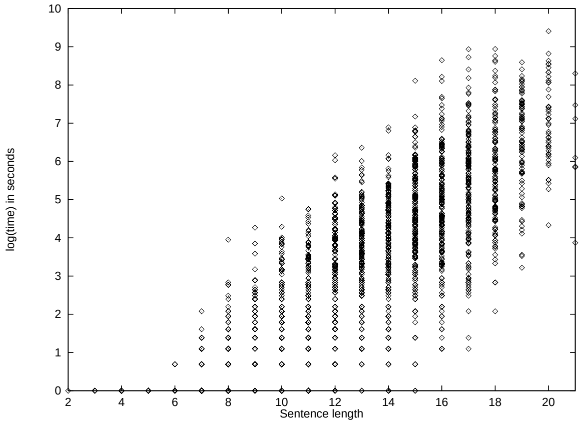


Figure A.3: Parse times plotted against sentence length. Coefficient of determination: $R^2 = 0.65$. (x-axis: Sentence length; y-axis: log(time in seconds))

Since we can easily determine the number of trees selected by a sentence before we start parsing, we can use this number to predict the number of edges that will be proposed by a parser when parsing this sentence, allowing us to better handle difficult cases *before* parsing.

We test the above hypothesis further by parsing the same set of sentences as above but this time using an oracle which tells us the correct elementary lexicalized structure for each word in the sentence. This eliminates lexical syntactic ambiguity but does not eliminate attachment ambiguity for the parser. The graph comparing the parsing times is shown in Figure A.5. As the comparison shows, the elimination of lexical ambiguity leads to a drastic increase in parsing efficiency. The total time taken to parse all 2250 sentences went from 548K seconds to 31.2 seconds.

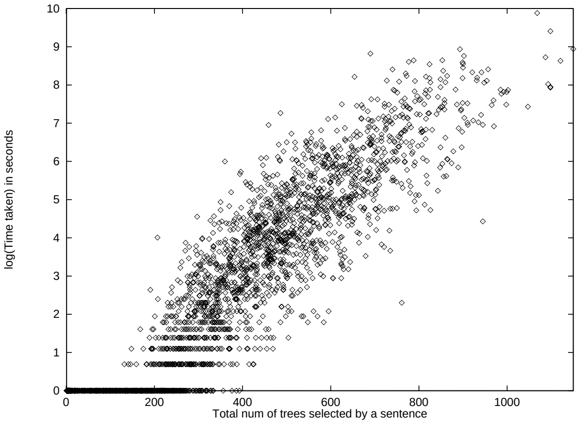


Figure A.4: The impact of syntactic lexical ambiguity on parsing times. Log of the time taken to parse a sentence plotted against the total number of trees selected by the sentence. Coefficient of determination: $R^2 = 0.82$. (x-axis: Total number of trees selected by a sentence; y-axis: log(time) in seconds).

Figure A.5 shows us that a model which disambiguates syntactic lexical ambiguity can potentially be extremely useful in terms of parsing efficiency. Thus disambiguation of tree assignment or SuperTagging [Srinivas, 1997b] of a sentence before parsing it might be a way of improving parsing efficiency. This gives us a way to reduce the parsing complexity for precisely the sentences which were problematic: the ones which selected too many trees. To test whether parsing times are reduced after SuperTagging we conducted an experiment in which the output of an n-best SuperTagger was taken as input to the parser. In our experiment we set n to be 60.[2] The time taken to parse the same set of sentences was again dramatically reduced (the total time taken was 21K seconds). However, the disadvantage of this method was that the coverage of the parser was reduced: 926 sentences (out of the 2250) did not get any parse. This was because some crucial tree was missing in the n-best output. The results are graphed in Figure A.6. The total number of derivations for all sentences went down to 1.01e+10 (the original total number was 1.4e+18) indicating (not surprisingly) that some attachment ambiguities persist although the number of trees are

[2] [Chen et al., 1999] shows that to get greater than 97% accuracy using SuperTagging the value of n must be quite high ($n > 40$). They use a different set of SuperTags and so we used their result simply to get an approximate estimate of the value of n.

reduced. We are experimenting with techniques where the output of the n-best SuperTagger is combined with other pieces of evidence to improve the coverage of the parser while retaining the speedup.

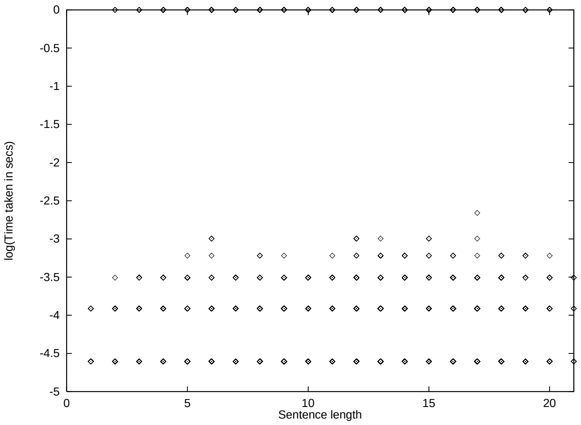


Figure A.5: Parse times when the parser gets the correct tree for each word in the sentence (eliminating any syntactic lexical ambiguity). The parsing times for all the 2250 sentences for all lengths never goes above 1 second. (x-axis: Sentence length; y-axis: log(time) in seconds)

A.0.3 Conclusion

In this chapter, we described an implementation of a chart-based head-corner parser for LTAGs. We ran some empirical tests by running the parser on 2250 sentences from the Wall Street Journal. We used a large Treebank Grammar to parse these sentences. We showed that the observed time complexity of the parser on these sentences does not increase predictably with longer sentence lengths. We presented evidence that indicates that the number of trees selected by the words in the sentence (a measure of the syntactic lexical ambiguity of a sentence) is a better predictor of complexity in LTAG parsing. We also showed how parsing time is dramatically affected by controlling the ambiguity of tree assignment to the words in the sentence.

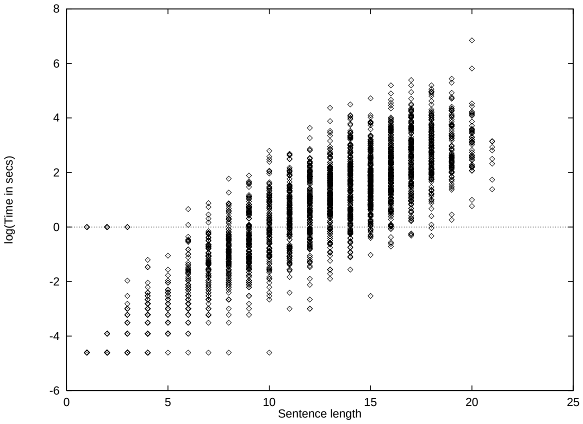


Figure A.6: Time taken by the parser after n-best SuperTagging ($n = 60$). (x-axis: Sentence length; y-axis: log(time) in seconds)

Appendix B

Computing Prefix Probabilities while Parsing

The problem of computing prefix probabilities for stochastic context-free languages has been discussed in [Jelinek and Lafferty, 1991, Stolcke, 1995]. The main idea leading to the solution of this problem is that all parts of context-free derivations that are potentially of unbounded size are captured into a set of equations that is solved "off-line", i.e. before a specific prefix is considered. These equations are then solved and the results are stored.

For computing the prefix probability for an actual input string, all possible derivations are considered and a probability is computed, but for certain parts of these derivations the results that were computed off-line are used, in such a way that the computation is guaranteed to terminate.

A case in point are the *unit rules*, i.e. rules of the form $A \rightarrow B$. Such rules potentially cause the grammar to be cyclic, which means that $A \rightarrow^* A$ for some nonterminal A, which allows certain strings to have derivations of unbounded size. However, also a rule of e.g. the form $A \rightarrow Ba$ may effectively behave like a unit rule if a contributes to the unknown suffix following the actual input that is considered as prefix.

For stochastic tree-adjoining grammars (STAGs) similar problems arise. STAGs that are well-behaved and allow a bounded number of derivations for each complete sentence may require an unbounded number of derivations to be considered once the input is regarded as prefix, followed by a suffix of unbounded length.

The key idea to solving this problem is again to break up derivations into parts that are of potentially unbounded size, and that are independent on actual input, and those that are always of bounded length, and that do depend on input symbols. The probabilities of the former subderivations can be computed off-line, and the results are combined with subderivations of the latter kind during computation of the prefix probability for a given string.

The distinction between the two kinds of subderivations requires a certain notational system that is difficult to define for tree-adjoining grammars. We will therefore concentrate on stochastic linear indexed grammars instead, relying on their strong equivalence to STAGs [Schabes, 1992] (see Section B.1).

Without loss of generality we require that rules are of the form $A[\eta \circ\circ] \rightarrow \alpha\ B[\eta' \circ\circ]\ \beta$, where $|\eta\eta'| = 1$ and α and β each consist of a string of nonterminals associated with empty stacks of indices, or of the form $A[\,] \rightarrow z$, where $|z| \leq 1$.

As usual, the arrow $\rightarrow$ is extended to be a binary relation between sentential forms, and its transitive and reflexive closure is denoted by $\rightarrow^*$. When we write $A[\sigma] \rightarrow^* \alpha\ B[\tau]\ \beta$ (or $A[\sigma] \rightarrow^* \alpha\ a\ \beta$) we mean that $B[\tau]$ (or a, respectively) is the distinguished descendent of $A[\sigma]$.

We first introduce a subrelation of $\rightarrow^*$ which is defined by $A[\sigma] \Rightarrow^* \epsilon$ if $A[\sigma] \rightarrow^* \epsilon$, and $A[\sigma] \Rightarrow^* B[\tau]$ if $A[\sigma] \rightarrow^* B[\tau]$, and this derivation does not end on a subderivation of the form $C[\,] \rightarrow^+ B[\,]$ (or more precisely $C[\tau] \rightarrow^+ B[\tau]$, where no elements that belong to τ are popped and pushed again), for any C. When we write $A[\sigma] \Rightarrow^* X$, then X is of the form $B[\tau]$ or ϵ.

Based on this, two further subrelations $\rightarrow^*_{ver}$ and $\rightarrow^*_{hor}$, are defined by means of deduction steps. The distinction between $\rightarrow^*_{ver}$ and $\rightarrow^*_{hor}$ is made in order to record how derivations were built up from subderivations. In the case of $\rightarrow^*_{hor}$, the derivation was constructed from two subderivations $A[\,] \rightarrow^* v\ B[\,]\ w$ and $B[\,] \rightarrow^* x\ C[\,]\ y$. In all other cases, we use $\rightarrow^*_{ver}$.

This distinction is needed to avoid spurious ambiguity in applications of the deduction steps: the result from combining $A[\,] \rightarrow^{*} v\ B[\,]\ w$ and $B[\,] \rightarrow^{*} x\ C[\,]\ y$, viz. $A[\,] \rightarrow^{*}_{hor} v\ x\ C[\,]\ y\ w$ is not allowed to combine with a third subderivation $C[\,] \rightarrow^{*} z\ D[\,]\ q$. Note that the desired derivation $A[\,] \rightarrow^{*}_{hor} v\ x\ z\ D[\,]\ q\ y\ w$ can be derived by combining $B[\,] \rightarrow^{*} x\ C[\,]\ y$ and $C[\,] \rightarrow^{*} z\ D[\,]\ q$. and then $A[\,] \rightarrow^{*} v\ B[\,]\ w$ with the result of this.

$$\frac{\qquad}{\epsilon \rightarrow^{*}_{ver} \epsilon} \tag{B.1}$$

$$\frac{A[\,] \rightarrow^{*}_{ver} v \quad \alpha \rightarrow^{*}_{ver} w \quad \alpha \neq \epsilon}{A[\,]\ \alpha \rightarrow^{*}_{ver} v\ w} \tag{B.2}$$

$$\frac{A[\,] \rightarrow^{*} a}{A[\,] \rightarrow^{*}_{ver} a} \tag{B.3}$$

$$\frac{\begin{array}{ll} A[\,] \rightarrow^{*} B[\sigma] & \alpha \rightarrow^{*}_{ver} v_{\alpha} \\ B[\circ\circ] \rightarrow \alpha\ C[p \circ\circ]\ \beta & \beta \rightarrow^{*}_{ver} v_{\beta} \\ C[\,] \rightarrow^{*} D[\,] & \gamma \rightarrow^{*}_{ver} v_{\gamma} \\ D[p \circ\circ] \rightarrow \gamma\ E[\circ\circ]\ \delta & \delta \rightarrow^{*}_{ver} v_{\delta} \\ E[\sigma] \Rightarrow^{*} X & v_{\alpha}v_{\beta}v_{\gamma}v_{\delta} \neq \epsilon \end{array}}{A[\,] \rightarrow^{*}_{ver} v_{\alpha}\ v_{\gamma}\ X\ v_{\delta}\ v_{\beta}} \tag{B.4}$$

$$\frac{\begin{array}{ll} A[\,] \rightarrow^{*} B[\sigma] & lab \in \{ver, hor\} \\ B[\circ\circ] \rightarrow \alpha\ C[p \circ\circ]\ \beta & \alpha \rightarrow^{*}_{ver} v_{\alpha} \\ C[\,] \rightarrow^{*}_{lab} v\ D[\sigma]\ w & \beta \rightarrow^{*}_{ver} v_{\beta} \\ D[\,] \rightarrow^{*} E[\,] & \gamma \rightarrow^{*}_{ver} v_{\gamma} \\ E[p \circ\circ] \rightarrow \gamma\ F[\circ\circ]\ \delta & \delta \rightarrow^{*}_{ver} v_{\delta} \\ F[\sigma] \Rightarrow^{*} X & v_{\alpha}v_{\beta}v_{\gamma}v_{\delta} \neq \epsilon \end{array}}{A[\,] \rightarrow^{*}_{ver} v_{\alpha}\ v\ v_{\gamma}\ X\ v_{\delta}\ w\ \ v_{\beta}} \tag{B.5}$$

$$\frac{\begin{array}{l} A[\,] \rightarrow^{*}_{ver} v\ B[\,]\ w \\ B[\,] \rightarrow^{*}_{lab} x\ C[\,]\ y \quad lab \in \{ver, hor\} \end{array}}{A[\,] \rightarrow^{*}_{hor} v\ x\ C[\,]\ y\ w} \tag{B.6}$$

$$\frac{\begin{array}{l} A[\,] \rightarrow^*_{ver} v\ B[\,]\ w \\ B[\,] \rightarrow^*_{ver} x \end{array}}{A[\,] \rightarrow^*_{ver} v\ x\ w} \tag{B.7}$$

$$\frac{\begin{array}{l} A[\,] \Rightarrow^* B[\sigma] \\ B[\,] \rightarrow^*_{hor} v\ C[\,]\ w \\ C[\sigma] \Rightarrow^* X \qquad \sigma \neq \epsilon \end{array}}{A[\,] \rightarrow^*_{ver} v\ X\ w} \tag{B.8}$$

For computing the inside probability of a given string w we apply the deduction steps in reverse for the derivation $S[\,] \rightarrow^*_{ver} w$, which gives rise to one or more partitionings into subderivations. We multiply the probabilities attached to the rules that are used in the derivations, and we add probabilities where more than one partitioning exists due to ambiguity.

We see that statements of the form $C[\,] \rightarrow^* D[\,]$ in e.g. step B.4 and $A[\,] \rightarrow^* a$ in step B.3 can themselves not be derived by the deduction steps. It is assumed the probabilities of such derivations are computed off-line, which is possible since they do not depend on actual input. Also the joint probability of the pair of derivations $A[\,] \rightarrow^* B[\sigma]$ and $E[\sigma] \Rightarrow^* X$ in step B.4 can be precomputed for a given combination of A, B, E, and X, even though there may be an infinite number of stacks σ.

It is easy to see that the backward application of the deduction steps must necessarily terminate. This is independent of whether a LIG allows infinite ambiguity.

If prefix probabilities are to be computed instead of inside probabilities, the deduction steps need to be slightly altered. For example, the condition $v_\alpha v_\beta v_\gamma v_\delta \neq \epsilon$ in step B.4 needs to be reformulated to the effect that at least one symbol from $v_\alpha v_\beta v_\gamma v_\delta$ should belong to the input, i.e. the prefix. Further, probabilities of derivations of the form $A[\,] \rightarrow^* B[\,]\ w$ should be computed off-line, where w belongs to the unknown suffix. (Cf. unit rules and rules of the form $A \rightarrow Ba$ in the case of context-free grammars.)

It is very easy to see that the deduction steps are consistent, in the sense that $\alpha \rightarrow^*_{ver} \beta$ or $\alpha \rightarrow^*_{hor} \beta$ implies $\alpha \rightarrow^* \beta$. That the deduction steps are also complete, i.e. that $A[\,] \rightarrow^*_{ver} w$ can be derived if $A[\,] \rightarrow^* w$, is more difficult to show and cannot be explained here due to length restrictions. The proof relies on a systematic way of partitioning a parse tree into smaller trees. Similarly, the proof that for any tree only one partitioning exists cannot be treated here.

B.0.4 Partioning derivations

In this optional appendix we explain how derivations are partitioned into subderivations. An important concept is that of *spines*. A spine is a path in the parse tree that leads from a node that is not a distinguished child of its father (or that does not have a father, in the case of the root), down to a leaf following distinguished children. This means that for an instance of a rule $A[\eta \circ\circ] \rightarrow \alpha\ B[\eta' \circ\circ]\ \beta$ in the parse tree the nodes corresponding to symbols in α and β are each the first node of a distinct spine, and the spine to which the node corresponding to $A[\eta \circ\circ]$ belongs leads down along the node corresponding to $B[\eta' \circ\circ]$.

At both ends of a spine, the stack of indices associated with the nonterminals is empty. In between, the height of the stack may alternately grow and shrink. This is shown in Figure B.1. The horizontal axis represents nodes along the spine, and the vertical axis represents the height of the stack.

At some instances of rules, non-empty input is found at some child of a node on the spine that does itself not belong to the spine. We always investigate such rules in pairs: if one rule pushes p on the stack, we locate the unique rule that pops that p; only one of the two rules needs to be associated with non-empty input. Three instances of such pairs of rules are indicated in the figure.

The part of the spine labelled by a and b are computed by applications of step B.4. From these two parts, the part labelled c is computed applying step B.6. This step combines paths in a “horizontal” way, hence the label *hor* in the consequent.

The path is extended to the path d in a vertical way by applying step B.8. Again vertically, step B.5 extends the path to path e by identifying one more pair of rules where non-empty input is found.

Each stack development along a spine, exemplified by Figure B.1, can be partitioned in exactly one way according to the deduction steps. In the full version of this paper we will give a sketch of the proof.

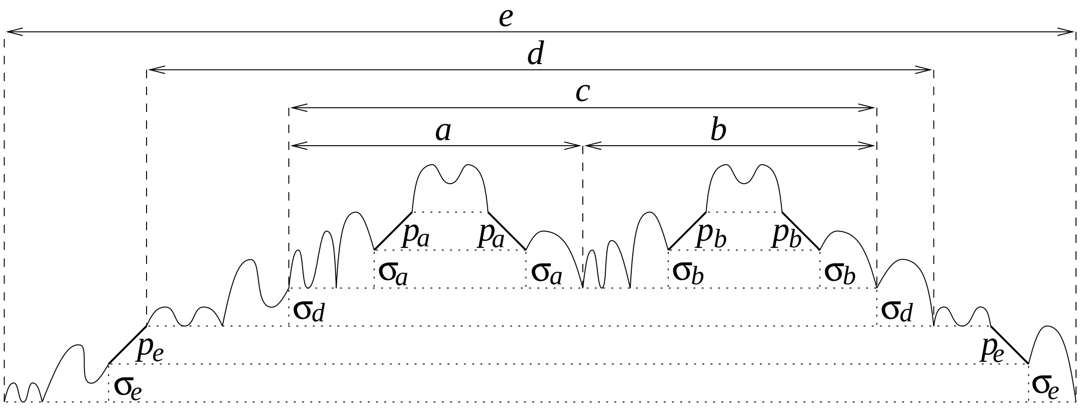


Figure B.1: Development of the stack along a spine, and partitioning according to deduction steps.

B.1 Embedding Probabilistic TAGs into LIGs

For any stochastic TAG we typically represent the moves of any algorithm that computes spans over the input string by using dotted rules. We can equivalently (and with considerable simplification in notation) denote the moves of such an algorithm in terms of a strongly equivalent LIG constructed from the given TAG.

The LIG is constructed as follows: the set of non-terminals are two symbols *top t* and *bottom b*; the set of terminals is the same as the TAG; the set of indices (stack symbols) is the set of nodes of the elementary trees. Finally, the set of rules (productions) are defined as follows, where we assume all elementary trees are binary branching and *spine* is defined as path from root to foot, and where p is the probability $P(rhs \mid lhs)$, \$ is the bottom of stack and [..] common between the lhs and a unique rhs nonterminal indicates that the stack is being passed between them:

1. η_0 is root of an initial tree α:

$$t[\$] \xrightarrow{p} t[\$\eta_0]$$

   where, p is the probability that α is the root of the derivation tree.

2. η is parent of η_1, η_2 and η_2 is on the spine:

$$b[..\eta] \xrightarrow{p=1} t[\$\eta_1]t[..\eta_2]$$

3. η is parent of η_1, η_2 and η_1 is on the spine:

$$b[..\eta] \xrightarrow{p=1} t[..\eta_1]t[\$\eta_2]$$

4. η is parent of η_1, η_2 and neither is on the spine:

$$b[..\eta] \xrightarrow{p=1} t[\$\eta_1]t[\$\eta_2]$$

5. η is a node where adjunction of a tree β with root η_r can occur:

$$\begin{aligned} t[..\eta] &\xrightarrow{p} b[..\eta] \\ t[..\eta] &\xrightarrow{p'} t[..\eta\eta_r] \end{aligned}$$

   where, $p = \phi(\eta \mapsto nil)$, and $p' = \phi(\eta \mapsto \beta)$.

6. η_f is a footnode:[1]

$$b[..\eta_f] \xrightarrow{p=1} b[..]$$

Let input string $w = a_1 \dots a_n$. Compute inside probabilities I^w bottom-up using the following equations. The equations can be easily converted into a dynamic programming algorithm using a CKY-style parsing algorithm.[2]

Init cases:

1. $b[\$\eta] \rightarrow a_{i+1}$, for $i = 0 \dots n$,

$$I(b, \eta, i, \_, \_, i+1) = \begin{cases} 1 & \text{if } b[\$\eta] \rightarrow a_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

[1] Or perhaps

$$b[..\eta\eta_f] \xrightarrow{p=1} b[..\eta]$$

[2] There is an exact correspondence between the LIG rules and the cases considered in the TAG CKY-style parser (see Vijay-Shanker's thesis).

2. $b[..\eta_f] \rightarrow b[..]$, for all footnodes η_f with $0 \leq i, j \leq n$, [3]

$$I(b, \eta_f, i, i, j, j) = 1$$

Inside Pr cases (with $l \leq n$):

1. $b[..\eta] \rightarrow t[..\eta_1]t[\$\eta_2]$,

$$I(b, \eta, i, j, k, l) = \sum_{m=k}^{l-1} I(t, \eta_1, i, j, k, m) \times I(t, \eta_2, m, \_, \_, l)$$

2. $b[..\eta] \rightarrow t[\$\eta_1]t[..\eta_2]$,

$$I(b, \eta, i, j, k, l) = \sum_{m=i+1}^{j} I(t, \eta_1, i, \_, \_, m) \times I(t, \eta_2, m, j, k, l)$$

3. $b[\$\eta] \rightarrow t[\$\eta_1]t[\$\eta_2]$,

$$I(b, \eta, i, \_, \_, l) = \sum_{m=i+1}^{l-1} I(t, \eta_1, i, \_, \_, m) \times I(t, \eta_2, m, \_, \_, l)$$

4. $t[..\eta] \rightarrow b[..\eta]$ and $t[..\eta] \rightarrow t[..\eta\eta_r]$, i.e. η is a node where adjunction of tree with root η_r can occur.

$$\begin{aligned} & I(t, \eta, i, j, k, l) = \\ & \quad I(b, \eta, i, j, k, l) \times P(t[..\eta] \rightarrow b[..\eta]) \\ & \quad + \sum_{r=i}^{j} \sum_{s=k}^{l} \sum_{\eta_r} I(t, \eta_r, i, r, s, l) \times I(b, \eta, r, j, k, s) \times P(t[..\eta] \rightarrow t[..\eta\eta_r]) \end{aligned}$$

[3] We can change this step from init to a predict step later. It might be simpler to just keep it as an init step, but point out that an implementation could change it to a predict step.

Appendix C

Learning TAG Lexicons from Corpora

In order to move towards more unsupervised techniques for statistical parsing we need methods that can acquire lexical knowledge from corpora. This information has to be acquired in a way that is straightforward to include within a statistical parser. In this chapter we describe two techniques which can be used to learn different parts of a Lexicalized Tree Adjoining Grammar from data. These are first steps towards a more general system for discovering various components of lexical knowledge which can be used to learn an LTAG from a minimally annotated or un-annotated corpus.

In Section C.1 we deal with the problem of learning subcategorization frames for verbs. In contrast with previous approaches we do not assume that we have a predefined set of subcat frames. Instead we discover the appropriate frames from data which is annotated with dependency structures. We report results on learning subcat frames using the Prague Dependency Treebank for Czech. The algorithm can be used to label dependents of a verb in the Czech treebank as either arguments or adjuncts. The method can also be applied to the output of a parser to bootstrap novel information about predicate-argument structure into a refined model which includes such information as part of the parser. Using our techniques, we are able to achieve 88% precision on unseen parsed text.

In Section C.2 we deal with the problem of discovering the underlying thematic structure of a verb where the syntactic context around the verb provide only ambiguous clues. This corresponds to the problem of learning verb diathesis classes, where each class corresponds to a tree family in a lexicalized Tree Adjoining Grammar. We describe a probabilistic classifier used to assign verbs to classes trained on a corpus that has been processed using a part-of-speech tagger and a phrase chunker.

C.1 Acquisition of Subcategorization Information from Corpora

The subcategorization of verbs is an essential issue in parsing, because it helps disambiguate the attachment of arguments and recover the correct predicate-argument relations by a parser. [Carroll and Minnen, 1998, Carroll and Rooth, 1998b] give several reasons why subcategorization information is important for a natural language parser. Machine-readable dictionaries are not comprehensive enough to provide this lexical information [Manning, 1993, Briscoe and Carroll, 1997]. Furthermore, such dictionaries are available only for very few languages. We need some general method for the automatic extraction of subcategorization information from text corpora.

Several techniques and results have been reported on learning subcategorization frames (SFs) from text corpora [Webster and Marcus, 1989, Brent, 1991, Brent, 1993, Brent, 1994, Ushioda et al., 1993, Manning, 1993, Ersan and Charniak, 1996, Briscoe and Carroll, 1997, Carroll and Minnen, 1998, Carroll and Rooth, 1998b]. All of this work deals with English. In this section we report on techniques that automatically extract SFs for Czech, which is a free word-order language, where verb complements have visible case marking.[1]

Apart from the choice of target language, this work also differs from previous work in other ways. Unlike all other previous work in this area, we do not assume that the set of SFs is known to us in advance. Also in contrast, we work with syntactically annotated data (the Prague Dependency Treebank, PDT [Hajič, 1998]) where the subcategorization information is *not* given; although this might be considered a simpler problem as compared

[1] One of the anonymous reviewers pointed out that [Basili and Vindigni, 1998] presents a corpus-driven acquisition of subcategorization frames for Italian.

to using raw text, we have discovered interesting problems that a user of a raw or tagged corpus is unlikely to face.

We first give a detailed description of the task of uncovering SFs and also point out those properties of Czech that have to be taken into account when searching for SFs. Then we discuss some differences from the other research efforts. We then present the three techniques that we use to learn SFs from the input data.

In the input data, many observed dependents of the verb are adjuncts. To treat this problem effectively, we describe a novel addition to the hypothesis testing technique that uses subset of observed frames to permit the learning algorithm to better distinguish arguments from adjuncts.

Using our techniques, we are able to achieve 88% precision in distinguishing arguments from adjuncts on unseen parsed text.

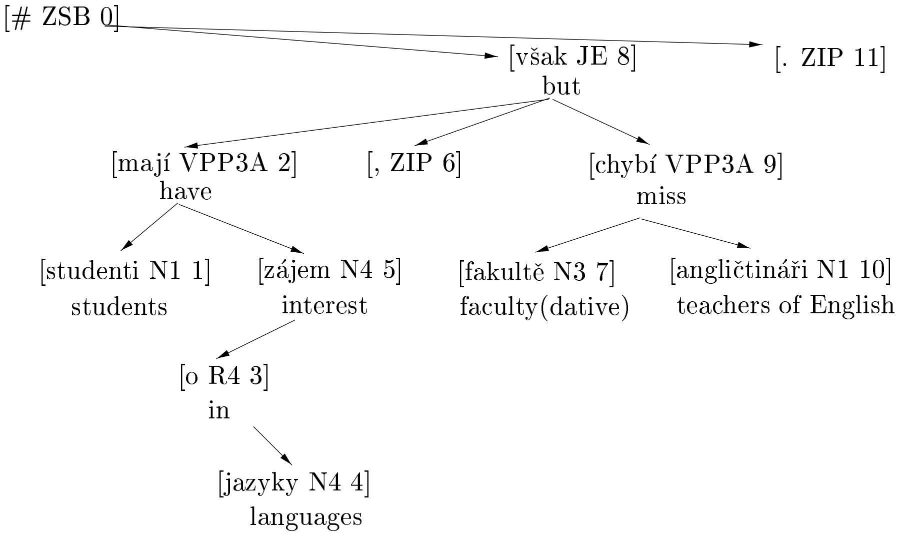
C.1.1 Task Description

In this section we describe precisely the proposed task. We also describe the input training material and the output produced by our algorithms.

Identifying subcategorization frames

In general, the problem of identifying subcategorization frames is to distinguish between arguments and adjuncts among the constituents modifying a verb. e.g., in "John saw Mary yesterday at the station", only "John" and "Mary" are required arguments while the other constituents are optional (adjuncts). There is some controversy as to the *correct* subcategorization of a given verb and linguists often disagree as to what is the right set of SFs for a given verb. A machine learning approach such as the one followed in this section sidesteps this issue altogether, since it is left to the algorithm to learn what is an appropriate SF for a verb.

Figure C.1 shows a sample input sentence from the PDT annotated with dependencies which is used as training material for the techniques described in this section. Each node in the tree contains a word, its part-of-speech tag (which includes morphological information) and its location in the sentence. We also use the functional tags which are part of the PDT annotation[2]. To make future discussion easier we define some terms here. Each daughter of a verb in the tree shown is called a *dependent* and the set of all dependents for that verb in that tree is called an *observed frame (OF)*. A *subcategorization frame (SF)* is a subset of the OF. For example the OF for the verb *mají (have)* in Figure C.1 is { *N1, N4* } and its SF is the same as its OF. Note that which OF (or which part of it) is a true SF is not marked in the training data. After training on such examples, the algorithm takes as input parsed text and labels each daughter of each verb as either an argument or an adjunct. It does this by selecting the most likely SF for that verb given its OF.



The students are interested in languages but the faculty is missing teachers of English.

Figure C.1: Example input to the algorithm from the Prague Dependency Treebank

Relevant properties of the Czech Data

Czech is a “free word-order” language. This means that the arguments of a verb do not have fixed positions and are not guaranteed to be in a particular configuration with respect to the verb.

The examples in (1) show that while Czech has a relatively free word-order some orders

[2] For those readers familiar with the PDT functional tags, it is important to note that the functional tag *Obj* does not always correspond to an argument. Similarly, the functional tag *Adv* does not always correspond to an adjunct. Approximately 50 verbs out of the total 2993 verbs require an adverbial argument.

are still marked. The SVO, OVS, and SOV orders in (1)a, (1)b, (1)c respectively, differ in emphasis but have the same predicate-argument structure. The examples (1)d, (1)e can only be interpreted as a question. Such word orders require proper intonation in speech, or a question mark in text.

The example (1)f demonstrates how morphology is important in identifying the arguments of the verb. cf. (1)f with (1)b. The ending *-a* of *Martin* is the only difference between the two sentences. It however changes the morphological case of *Martin* and turns it from subject into object. Czech has 7 cases that can be distinguished morphologically.

(1) a. Martin otvírá soubor. (SVO: Martin opens the file)
b. Soubor otvírá Martin. (OVS: ≠ the file opens Martin)
c. Martin soubor otvírá.
d. #Otvírá Martin soubor.
e. #Otvírá soubor Martin.
f. Soubor otvírá Martina. (= the file opens Martin)

Almost all the existing techniques for extracting SFs exploit the relatively fixed word-order of English to collect features for their learning algorithms using fixed patterns or rules (see Table C.2 for more details). Such a technique is not easily transported into a new language like Czech. Fully parsed training data can help here by supplying all dependents of a verb. The observed frames obtained this way have to be *normalized* with respect to the word order, e.g. by using an alphabetic ordering.

For extracting SFs, prepositions in Czech have to be handled carefully. In some SFs, a particular preposition is required by the verb, while in other cases it is a class of prepositions such as locative prepositions (e.g. *in, on, behind, ...*) that are required by the verb. In contrast, adjuncts can use a wider variety of prepositions. Prepositions specify the case of their noun phrase complements but a preposition can take complements with more than one case marking with a different meaning for each case. (e.g. *na mostě* = *on the bridge; na most* = *onto the bridge*). In general, verbs select not only for particular prepositions but also indicate the case marking for their noun phrase complements.

Argument types

We use the following set of labels as possible arguments for a verb in our corpus. They are derived from morphological tags and simplified from the original PDT definition [Hajič and Hladká, 1998, Hajič, 1998]; the numeric attributes are the case marking identifiers. For prepositions and clause complementizers, we also save the lemma in parentheses.

- Noun phrases: N4, N3, N2, N7, N1
- Prepositional phrases: R2(bez), R3(k), R4(na), R6(na), R7(s), ...
- Reflexive pronouns *se*, *si*: PR4, PR3
- Clauses: S, JS(že), JS(zda)
- Infinitives (VINF)
- passive participles (VPAS)
- adverbs (DB)

We do not specify which SFs are possible since we aim to discover these (see Section C.1.1).

C.1.2 Three methods for identifying subcategorization frames

We describe three methods that take as input a list of verbs and associated observed frames from the training data (see Section C.1.1), and learn an association between verbs and possible SFs. We describe three methods that arrive at a numerical score for this association.

However, before we can apply any statistical methods to the training data, there is one aspect of using a treebank as input that has to be dealt with. A correct frame (verb + its arguments) is almost always accompanied by one or more adjuncts in a real sentence. Thus the *observed frame* will almost always contain noise. The approach offered by Brent and others counts all observed frames and then decides which of them do not associate strongly with a given verb. In our situation this approach will fail for most of the observed frames because we rarely see the correct frames isolated in the training data. For example, from the occurrences of the transitive verb *absolvovat* ("go through something") that occurred ten times in the corpus, no occurrence consisted of the verb-object pair alone. In other words, the correct SF constituted 0% of the observed situations. Nevertheless, for each observed frame, one of its subsets was the correct frame we sought for. Therefore, we considered all possible subsets of all observed frames. We used a technique which steps through the subsets of each observed frame from larger to smaller ones and records their frequency in data. Large infrequent subsets are suspected to contain adjuncts, so we replace them by more frequent smaller subsets. Small infrequent subsets may have elided some arguments and are rejected. Further details of this process are discussed in Section C.1.2.

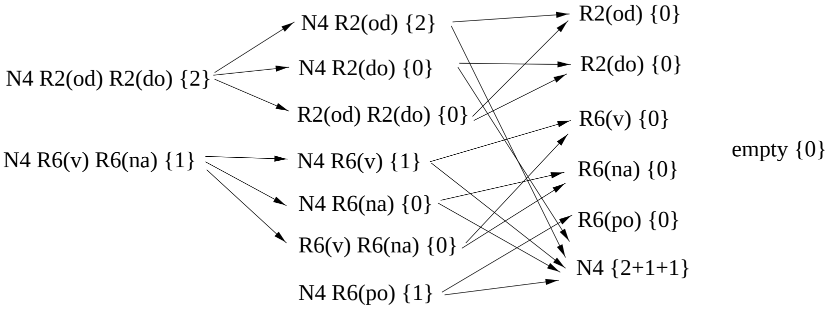


Figure C.2: Computing the subsets of observed frames for the verb *absolvovat*. The counts for each frame are given within braces {}. In this example, the frames *N4 R2(od), N4 R6(v)* and *N4 R6(po)* have been observed with other verbs in the corpus. Note that the counts in this figure do not correspond to the real counts for the verb *absolvovat* in the training corpus.

The methods we present here have a common structure. For each verb, we need to associate a score to the hypothesis that a particular set of dependents of the verb are arguments of that verb. In other words, we need to assign a value to the hypothesis that the observed frame under consideration is the verb's SF. Intuitively, we either want to test for independence of the observed frame and verb distributions in the data, or we want to test how likely is a frame to be observed with a particular verb without being a valid SF. We develop these intuitions with the following well-known statistical methods. For further background on these methods the reader is referred to [Bickel and Doksum, 1977, Dunning, 1993].

Likelihood ratio test

Let us take the hypothesis that the distribution of an observed frame f in the training data is independent of the distribution of a verb v. We can phrase this hypothesis as $p(f \mid v) = p(f \mid !v) = p(f)$, that is distribution of a frame f given that a verb v is present is the same as the distribution of f given that v is not present (written as $!v$). We use the log likelihood test statistic [Bickel and Doksum, 1977](p.209) as a measure to discover particular frames and verbs that are highly associated in the training data.

$$\begin{aligned} k_1 &= c(f,v) \\ n_1 &= c(v) = c(f,v) + c(!f,v) \\ k_2 &= c(f,!v) \\ n_2 &= c(!v) = c(f,!v) + c(!f,!v) \end{aligned}$$

where $c(\cdot)$ are counts in the training data. Using the values computed above:

$$\begin{aligned} p_1 &= \frac{k_1}{n_1} \\ p_2 &= \frac{k_2}{n_2} \\ p &= \frac{k_1 + k_2}{n_1 + n_2} \end{aligned}$$

Taking these probabilities to be binomially distributed, the log likelihood statistic [Dunning, 1993] is given by:

$$\begin{gathered} -2\log\lambda = \\ 2[\log L(p_1,k_1,n_1) + \log L(p_2,k_2,n_2) - \\ \log L(p,k_1,n_2) - \log L(p,k_2,n_2)] \end{gathered}$$

where,

$$\log L(p, n, k) = k \log p + (n - k) \log(1 - p)$$

According to this statistic, the greater the value of $-2 \log \lambda$ for a particular pair of observed frame and verb, the more likely that frame is to be valid SF of the verb.

T-scores

Another statistic that has been used for hypothesis testing is the *t-score*. Using the definitions from Section C.1.2 we can compute t-scores using the equation below and use its value to measure the association between a verb and a frame observed with it.

$$T = \frac{p_1 - p_2}{\sqrt{\sigma^2(n_1, p_1) + \sigma^2(n_2, p_2)}}$$

where,

$$\sigma(n, p) = np(1 - p)$$

In particular, the hypothesis being tested using the t-score is whether the distributions p_1 and p_2 are *not* independent. If the value of T is greater than some threshold then the verb v should take the frame f as a SF.

Binomial Models of Miscue Probabilities

Once again assuming that the data is binomially distributed, we can look for frames that co-occur with a verb by exploiting the miscue probability: the probability of a frame co-occuring with a verb when it is not a valid SF. This is the method used by several earlier sections on SF extraction starting with [Brent, 1991, Brent, 1993, Brent, 1994].

Let us consider probability $p_{!f}$ which is the probability that a given verb is observed with a frame but this frame is not a valid SF for this verb. $p_{!f}$ is the error probability on identifying a SF for a verb. Let us consider a verb v which does *not* have as one of its valid SFs the frame f. How likely is it that v will be seen m or more times in the training data with frame f? If v has been seen a total of n times in the data, then $H^*(p_{!f}; m, n)$ gives us this likelihood.

$$H^*(p_{!f}; m, n) = \sum_{i=m}^{n} p_{!f}^{i}(1 - p_{!f})^{n-i} \left(\begin{array}{c} n \\ i \end{array} \right)$$

If $H^*(p; m, n)$ is less than or equal to some small threshold value then it is extremely unlikely that the hypothesis is true, and hence the frame f must be a SF of the verb v. Setting the threshold value to 0.05 gives us a 95% or better confidence value that the verb v has been observed often enough with a frame f for it to be a valid SF.

Initially, we consider only the observed frames (OFs) from the treebank. There is a chance that some are subsets of some others but now we count only the cases when the OFs were seen themselves. Let's assume the test statistic rejected the frame. Then it is not a real SF but there probably is a subset of it that is a real SF. So we select exactly one of the subsets whose length is one member less: this is the *successor* of the rejected frame and inherits its frequency. Of course one frame may be successor of several longer frames and it can have its own count as OF. This is how frequencies accumulate and frames become more likely to survive. The example shown in Figure C.2 illustrates how the subsets and successors are selected.

An important point is the selection of the successor. We have to select only one of the n possible successors of a frame of length n, otherwise we would break the total frequency of the verb. Suppose there is m rejected frames of length n. This yields $m * n$ possible modifications to consider before selection of the successor. We implemented two methods for choosing a single successor frame:

1. Choose the one that results in the strongest preference for some frame (that is, the successor frame results in the lowest entropy across the corpus). This measure is sensitive to the frequency of this frame in the rest of corpus.

2. Random selection of the successor frame from the alternatives.

Random selection resulted in better precision (88% instead of 86%). It is not clear why a method that is sensitive to the frequency of each proposed successor frame does not perform better than random selection.

The technique described here may sometimes result in subset of a correct SF, discarding one or more of its members. Such frame can still help parsers because they can at least look for the dependents that have survived.

C.1.3 Evaluation

For the evaluation of the methods described above we used the Prague Dependency Treebank (PDT). We used 19,126 sentences of training data from the PDT (about 300K words). In this training set, there were 33,641 verb tokens with 2,993 verb types. There were a total of 28,765 *observed frames* (see Section C.1.1 for explanation of these terms). There were 914 verb types seen 5 or more times.

Since there is no electronic valence dictionary for Czech, we evaluated our filtering technique on a set of 500 test sentences which were unseen and separate from the training data. These test sentences were used as a gold standard by distinguishing the arguments and adjuncts manually. We then compared the accuracy of our output set of items marked as either arguments or adjuncts against this gold standard.

First we describe the baseline methods. Baseline method 1: consider each dependent of a verb an adjunct. Baseline method 2: use just the longest known observed frame matching the test pattern. If no matching OF is known, find the longest partial match in the OFs seen in the training data. We exploit the functional and morphological tags while matching. No statistical filtering is applied in either baseline method.

A comparison between all three methods that were proposed in this section is shown in Table C.1.

The experiments showed that the method improved precision of this distinction from 57% to 88%. We were able to classify as many as 914 verbs which is a number outperformed only by Manning, with 10x more data (note that our results are for a different language).

Also, our method discovered 137 subcategorization frames from the data. The known upper bound of frames that the algorithm could have found (the total number of the *observed frame* types) was 450.

	Baseline 1	Baseline 2	Lik. Ratio	T-scores	Hyp. Testing
Precision	55%	78%	82%	82%	88%
Recall:	55%	73%	77%	77%	74%
$F_{\beta=1}$	55%	75%	79%	79%	80%
% unknown	0%	6%	6%	6%	16%
Total verb nodes	1027	1027	1027	1027	1027
Total complements	2144	2144	2144	2144	2144
Nodes with known verbs	1027	981	981	981	907
Complements of known verbs	2144	2010	2010	2010	1812
Correct Suggestions	1187.5	1573.5	1642.5	1652.9	1596.5
True Arguments	956.5	910.5	910.5	910.5	834.5
Suggested Arguments	0	1122	974	1026	674
Incorrect arg suggestions	0	324	215.5	236.3	27.5
Incorrect adj suggestions	956.5	112.5	152	120.8	188

Table C.1: Comparison between the baseline methods and the three methods proposed in this section. Some of the values are not integers since for some difficult cases in the test data, the value for each argument/adjunct decision was set to a value between $[0,1]$. *Recall* is computed as the number of known verb complements divided by the total number of complements. *Precision* is computed as the number of correct suggestions divided by the number of known verb complements. $F_{\beta=1} = (2 \times p \times r)/(p+r)$. *% unknown* represents the percent of test data not considered by a particular method.

C.1.4 Comparison with related work

Preliminary work on SF extraction from corpora was done by [Brent, 1991, Brent, 1993, Brent, 1994] and [Webster and Marcus, 1989, Ushioda et al., 1993]. Brent [Brent, 1993, Brent, 1994] uses the standard method of testing miscue probabilities for filtering frames observed with a verb. [Brent, 1994] presents a method for estimating $p_{!f}$. Brent applied his method to a small number of verbs and associated SF types. [Manning, 1993] applies Brent's method to parsed data and obtains a subcategorization dictionary for a larger set of verbs. [Briscoe and Carroll, 1997, Carroll and Minnen, 1998] differs from earlier work in that a substantially larger set of SF types are considered; [Carroll and Rooth, 1998b] use an EM algorithm to learn subcategorization as a result of learning rule probabilities, and, in turn, to improve parsing accuracy by applying the verb SFs obtained. [Basili and Vindigni, 1998] use a conceptual clustering algorithm for acquiring subcategorization frames for Italian. They establish a partial order on partially overlapping OFs (similar to our OF subsets) which is then used to suggest a potential SF. A complete comparison of all the previous approaches with the current work is given in Table C.2.

While these approaches differ in size and quality of training data, number of SF types (e.g. intransitive verbs, transitive verbs) and number of verbs processed, there are properties that all have in common. They all assume that they know the set of possible SF types in advance. Their task can be viewed as assigning one or more of the (known) SF types to a given verb. In addition, except for [Briscoe and Carroll, 1997, Carroll and Minnen, 1998], only a small number of SF types is considered.

Using a dependency treebank as input to our learning algorithm has both advantages and drawbacks. There are two main advantages of using a treebank:

- Access to more accurate data. Data is less noisy when compared with tagged or parsed input data. We can expect correct identification of verbs and their dependents.

- We can explore techniques (as we have done in this section) that try and learn the set of SFs from the data itself, unlike other approaches where the set of SFs have to be set in advance.

Also, by using a treebank we can use verbs in different contexts which are problematic for previous approaches, e.g. we can use verbs that appear in relative clauses. However, there are two main drawbacks:

- Treebanks are expensive to build and so the techniques presented here have to work with less data.
- All the dependents of each verb are visible to the learning algorithm. This is contrasted with previous techniques that rely on finite-state extraction rules which ignore many dependents of the verb. Thus our technique has to deal with a different kind of data as compared to previous approaches.

We tackle the second problem by using the method of observed frame subsets described in Section C.1.2.

Previous work	Data	#SFs	#verbs tested	Method	Miscue rate	Corpus
[Ushioda et al., 1993]	POS + FS rules	6	33	heuristics	NA	WSJ (300K)
[Brent, 1993]	raw + FS rules	6	193	Hypothesis testing	iterative estimation	Brown (1.1M
[Manning, 1993]	POS + FS rules	19	3104	Hypothesis testing	hand	NYT (4.1M)
[Brent, 1994]	raw + heuristics	12	126	Hypothesis testing	non-iter estimation	CHILDES (3
[Ersan and Charniak, 1996]	Full parsing	16	30	Hypothesis testing	hand	WSJ (36M)
[Briscoe and Carroll, 1997]	Full parsing	160	14	Hypothesis testing	Dictionary estimation	various (70K)
[Carroll and Rooth, 1998b]	Unlabeled	9+	3	Inside-outside	NA	BNC (5-30M)
Current Work	Fully Parsed	Learned 137	914	Subsets+ Hyp. testing	Estimate	PDT (300K)

Table C.2: Comparison with previous work on automatic SF extraction from corpora

C.1.5 Conclusion

We are currently incorporating the SF information produced by the methods described in this section into a parser for Czech. We hope to duplicate the increase in performance shown by treebank-based parsers for English when they use SF information. Our methods can also be applied to improve the annotations in the original treebank that we use as training data. The automatic addition of subcategorization to the treebank can be exploited to add predicate-argument information to the treebank.

Also, techniques for extracting SF information from data can be used along with other research which aims to discover relationships between different SFs of a verb [Stevenson and Merlo, 1999, Lapata and Brew, 1999, Lapata, 1999, Stevenson et al., 1999].

The statistical models in this section were based on the assumption that given a verb, different SFs occur independently. This assumption is used to justify the use of the binomial. Future work perhaps should look towards removing this assumption by modeling the dependence between different SFs for the same verb using a multinomial distribution.

To summarize: we have presented techniques that can be used to learn subcategorization information for verbs. We exploit a dependency treebank to learn this information, and moreover we discover the final set of valid subcategorization frames from the training data. We achieve upto 88% precision on unseen data.

We have also tried our methods on data which was automatically morphologically tagged which allowed us to use more data (82K sentences instead of 19K). The performance went up to 89% (a 1% improvement).

C.2 Learning Verb Classes from Corpora

In this section we report on some experiments in the classification of verbs based on their underlying thematic structure. The objective of the classification is to correctly identify verbs that take the same number and category of arguments but assign different thematic roles to these arguments. In the literature, this is termed as the classification of verb diathesis roles. We exploit some ideas that have proposed in the literature, but use minimally annotated data to construct our classifier. The data we use is only passed through a part-of-speech tagger and a chunker which is used to identify base phrasal categories to identify potential arguments of each verb.

C.2.1 Verb classes and LTAG Trees

In an LTAG grammar, the shape of the elementary trees assigned to each predicate is determined by the arguments taken by that predicate. For example, the verb *eat* will take an entire family of trees associated with its predicate-argument structure of *NP0 V NP1*, where *NP0* and *NP1* represent the arguments of the predicate *eat*. However, in some case the number and category of the arguments are identical, but the elementary trees still have to be distinguished. In Figure C.3 T1 is an elementary tree that takes two NP arguments while T2 and T3 take a single NP to the left of the verb. However T2 and T3 express a different role for the argument NP with respect to the verb.

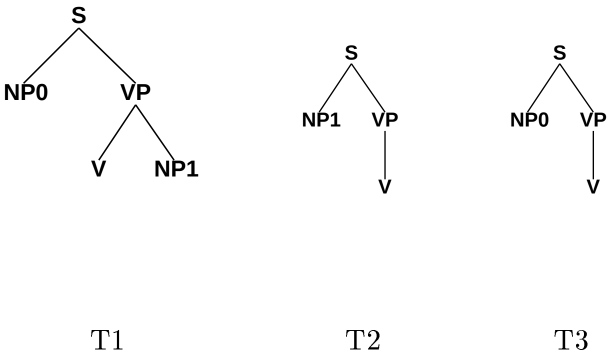


Figure C.3: TAG elementary trees and verb classification.

Given 3 TAG trees above, if we wanted to lexicalize the verb *broke* with one or more of these trees then by examining the various syntactic contexts in which it can occur we can into the class represented by a subset of these trees. We know that *broke* can occur in the following sentence:

(2) a. [John]/NP0 [broke]/V0 [the window]/NP1

Therefore, broke must lexicalize T1. We also know that *broke* can occur in the following sentence:

(3) a. [The window]/NP1 [broke]/VP0

This use of the verb *broke* is related to the following meaning:

(4) a. [The window]/NP1 [was broken]/V0 by [John]/NP0

Therefore, *broke* also lexicalizes T2.

Knowing the observed syntactic contexts in which *break* (in its various morphological forms) can occur, we can classify it into the class represented by a set of T1 and T2. In the remainder of this section we will look at the problem of classifying verbs in such classes automatically.

We create a probabilistic classifier that can automatically classify a set of verbs with a reasonable error rate. We use the hypothesis proposed by [Stevenson and Merlo, 1999] that although a verb in a particular class can occur in all of the syntactic contexts as verbs from other classes the statistical distributions can be distinguished. In other words, verbs from certain classes will be more likely to occur in some syntactic contexts than others. We identify features that pick out the verb occurences in these contexts. By using these features, we will attempt to determine the classification of those verbs. In this experiment, we will consider the following set of classes: unergative, unaccusative, object- drop, and transitive.

C.2.2 Experimental Setup

We are trying to construct the probabilistic classifier that can classify verbs into 4 lexical classes: unergative, unaccusative, object-drop, and transitive. The properties of verbs belong to these classes are described below:

An unergative verb is an intransitive action verb whose transitive form is the causative counter-part of the intransitive form. For example,

(5) a. The horse raced past the castle.

b. That knight raced the horse past the castle.

An unaccusative verb is similar to an unergative verb. It has a transitive form, which is the causative counter part of intransitive form. However, it is an intransitive change of state verb.

(6) a. The butter melted in the pan.

b. My mom melted the butter in the pan.

An object-drop verb is a verb that has a non-causative transitive/intransitive alternation. The object of a sentence is optional. For example,

(7) a. She ate lunch quickly.
b. She ate quickly.

A (pure) transitive verb is a verb that can occur in sentences only in transitive form. For example,

(8) a. John rejected Jane's proposal.

An unergative verb and an accusative verb have the causative alternation; an object of transitive sentence can become a subject of intransitive sentence. An object-drop verb does not have an alternation, but it can leave the object of the sentence. A transitive verb does not have any property above. Also, it can only occur in the transitive form, while the rest can occur in both transitive and intransitive forms. In our experiment, we will test 76 verbs that are in one of these four classes.

C.2.3 Steps in Constructing the Classifier

To construct the classifier, we will identify features that can be used to accurately distinguish verbs into different classes. We use two learning algorithms: decision trees and AdaBoost to construct the classifier. The features are computed to be the probability of observing a particular feature with each verb to be classified. We use C5.0 to generate the tree. The features are extracted from a 23M word corpus of WSJ text (LDC WSJ 1988 collection).

We prepare the corpus by passing it through Adwait Ratnaparkhi's part-of-speech tagger and then running Steve Abney's chunker over the entire text. The output of this stage and the input to our feature extractor is shown below.

Pierre	NNP	nx	2
Vinken	NNP		
,	,		
61	CD	ax	3
years	NNS		
old	JJ		
,	,		
will	MD	vx	2
join	VB		
the	DT	nx	2
board	NN		
as	IN		
a	DT	nx	3
nonexecutive	JJ		
director	NN		
Nov.	NNP		
29	CD		
.	.		

We use the following features to construct the classifier:

1. simple past (VBD), and past participle(VBN)

2. active (ACT) and passive (PASS)

3. transitive (TRAN) and intransitive (INTRAN)

4. causative (CAUS)

5. animacy (ANIM)

6. Part of Speech of Subject Phrase and Object Phrase

To calculate all the probability values of each features, we perform the following steps:

1. Finding the main verb of the sentences

   To find the main verb, we constructed a deterministic finite-state automaton that finds the main verb withing the verb phrase chunks (marked by vx).

2. Obtaining the frequency distribution of the features

   To find the frequency distribution of each feature of the verb, we used the following formula:

   $$P(V_j) = \frac{C(V_j)}{\sum_{1 \leq x \leq N} C(V_x)}$$

   where $P(V_j)$ is the distribution of feature j of the verb, and $C(V_j)$ is the number of times this feature of the verb was observed in the corpus. The features computed using this formula are: ACT, PASS, TRAN, INTRAN, VBD, and VBN.

3. The causative feature: CAUS

   To correctly obtain the causative values of the testing verbs, we needed to know the meaning of the sentences. In this experiment, We could only approximate the value by using the following approach. Also, the causative value is not probabilistic (as above) but heuristic.

   We extract the subjects and objects of verbs and put them into two sets. We use the last noun of the subject noun phrase and object noun phrase (tagged by NN, NNS, NNP, or NNPS), as the subject and object of the sentences. Then the causative value is

   CAUS = overlap/sum of all subject and objects in multiset

   where the overlap is defined as the largest multiset of elements belonging to both subjects and objects multisets.

   if subject is in the set $\{a, a, b, c\}$ and object is in set $\{a, d\}$, the intersection between both set will be $\{a, a\}$, and the causative value will be $\frac{2}{(4+2)} = \frac{1}{3}$.

   if subject is in the set $\{a, a, b, c\}$ and object is in the set $\{a, b, d\}$, the intersection between both set will be $\{a, a, b\}$, and the causative value will be $\frac{(2+1)}{(4+3)} = \frac{3}{7}$.

4. The animate feature: ANIM

   Similar to CAUS, we can only approximate the value of animacy. We use the the following formula to find the value:

   ANIM = number of occurrence of pronoun in subject/number of occurrence of verbs

   The set of pronouns used are *I*, *we*, *you*, *she*, *he*, and *they*.

5. Part of Speech of object and subject

   We count the occurrence of the each last part of subject noun phrase and object noun phrase. Then, we find the frequency distribution by using the same formula as before:

$$P(V_j) = \frac{C(V_j)}{\sum_{1 \le x \le N} C(V_x)}$$

   Where $P(V_j)$ is the distribution of part of speech j, and $C(V_j)$ is the number of occurrences of part of speech j. Also, we limit the part of speech to only the following tags of speech: NNP, NNPS, EX, PRP, and SUCH, where NNP is singular noun phrase, NNPS is plural noun phrase, EX is there, PRP is personal pronoun, and SUCH is such.

6. Construct the Classifier with C5.0

   After we obtain all the probabilistic distributions of the features of our testing verbs, we then use C5.0 to construct the classifier, which uses AdaBoost (over decision trees) and rule set induction. We use 10 fold cross-validation when computing the results.

C.2.4 Results

We have tried to add many combinations of features and then see whether the new features we add contribute to the reduction of error rate or not. The following are the results of the various feature combinations.

The results obtained from using decision trees vs. AdaBoost do not differ that much (except the first case). So we will only report the error rate obtained by the decision tree.

Features	Average error rate from Decision Tree	SE	Average error rate from Rule Set	SE
TRAN, INTRAN, VBD, VBN, PASS, ACT	49.4%	1.1%	67.7%	0.9%
TRAN, INTRAN, VBD, VBN, PASS, ACT, CAUS	41.1%	0.8%	40.8%	0.6%
TRAN, INTRAN, VBD, VBN, PASS, ACT, ANIM	37.5%	0.8%	36.9%	1.0%
TRAN, INTRAN, VBD, VBN, PASS, ACT, PART OF SPEECH	39.2%	0.8%	38.1%	1.1%
TRAN, INTRAN, VBD, VBN, PASS, ACT, CAUS, ANIM	**33.4%**	**0.7%**	**33.9%**	**0.8%**
TRAN, INTRAN, VBD, VBN, PASS, ACT, CAUS, PART OF SPEECH	39.0%	0.7%	37.1%	0.9%
TRAN, INTRAN, VBD, VBN, PASS, ACT, ANIM, PART OF SPEECH	35.8%	1.3%	35.9%	1.7%
TRAN, INTRAN, VBD, VBN, PASS, ACT, CAUS, ANIM, PART OF SPEECH	39.5%	1.0%	38.3%	1.0%

Figure C.4: Results of the verb classification

With our base features, ACT, PASS, VBD, VBN, TRAN, and INTRAN along with 10 fold cross-validation and adaptive boosting, we get the average error rate of 49.4%. We can see that when we add the CAUS feature, the average error decrease to 41.1%. So the CAUS help decreasing the error rate. Also, when we add the ANIM feature, we get a much better performance. Our average error rate decreases to 37.5%. This is the lowest error rate we can achieve by adding one extra feature aside the base features. So, ANIM feature is an important feature that we can use to construct the classifier. When we add PART OF SPEECH feature, the error rate also decreases to 39.2%. Therefore, the PART OF SPEECH also helps reduce the error rate as well. When we put together the CAUS feature and ANIM feature, we achieve the lowest error rate, which is 33.4%. When we put the PART OF SPEECH and CAUS features together, the error rate does not really decrease (39.0%), comparing to the result with only PART OF SPEECH feature. The reason of this result should be that there are some parts of PART OF SPEECH feature and CAUS feature that overlap. When we add ANIM and PART OF SPEECH features together, the error rate does decrease to 35.8%. Although the result is not as good as result of using ANIM and CAUS features, the combination of ANIM and PART OF SPEECH features are still considered effective features that we can use to construct the classifier. We then combine all the features together. The result as expected is not very good. The error rate is 39.5%. The reason should be the same reason of the combination of CAUS and PART OF THE SPEECH features. That is there are some overlap between these two features.

C.2.5 Conclusion

In this experiment, we automatically classified a set of verbs on distributions of some selected features extracted from a 23M word WSJ corpus. We used c5.0 to construct the classifier using values of those features. The result we obtained is promising we were able to construct a classifier that has an error rate of 33.4%.

Bibliography

[Basili and Vindigni, 1998] Basili, R. and Vindigni, M. (1998). Adapting a subcategorization lexicon to a domain. In *Proceedings of the ECML'98 Workshop* TANLPS: Towards adaptive NLP-driven systems: linguistic information, learning methods and applications, Chemnitz, Germany.

[Bickel and Doksum, 1977] Bickel, P. and Doksum, K. (1977). *Mathematical Statistics.* Holden-Day Inc.

[Blum and Mitchell, 1998] Blum, A. and Mitchell, T. (1998). Combining Labeled and Unlabeled Data with Co-Training. In *In Proc. of 11th Annual Conf. on Comp. Learning Theory (COLT)*, pages 92–100.

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

[Brent, 1991] Brent, M. (1991). Automatic acquisition of subcategorization frames from untagged text. In *Proceedings of the 29th Meeting of the ACL*, pages 209–214, Berkeley, CA.

[Brent, 1993] Brent, M. (1993). From grammar to lexicon: unsupervised learning of lexical syntax. *Computational Linguistics*, 19(3):243–262.

[Brent, 1994] Brent, M. (1994). Acquisition of subcategorization frames using aggregated evidence from local syntactic cues. *Lingua*, 92:433–470. Reprinted in Acquisition of the Lexicon, L. Gleitman and B. Landau (Eds.). MIT Press, Cambridge, MA.

[Brill, 1997] Brill, E. (1997). Unsupervised learning of disambiguation rules for part of speech tagging. In *Natural Language Processing Using Very Large Corpora*. Kluwer Academic Press.

[Briscoe and Carroll, 1997] Briscoe, T. and Carroll, J. (1997). Automatic extraction of subcategorization from corpora. In *Proceedings of the 5th ANLP Conference*, pages 356–363, Washington, D.C. ACL.

[Carroll and Rooth, 1998a] Carroll, G. and Rooth, M. (1998a). Valence Induction with a Head-Lexicalized PCFG. `http://xxx.lanl.gov/abs/cmp-lg/9805001`.

[Carroll and Rooth, 1998b] Carroll, G. and Rooth, M. (1998b). Valence induction with a head-lexicalized PCFG. In *Proceedings of the 3rd Conference on Empirical Methods in Natural Language Processing (EMNLP 3)*, Granada, Spain.

[Carroll and Minnen, 1998] Carroll, J. and Minnen, G. (1998). Can subcategorisation probabilities help a statistical parser. In *Proceedings of the 6th ACL/SIGDAT Workshop on Very Large Corpora (WVLC-6)*, Montreal, Canada.

[Carroll and Weir, 1997] Carroll, J. and Weir, D. (1997). Encoding frequency information in lexicalized grammars. In *Proc. 5th Int'l Workshop on Parsing Technologies IWPT-97*, Cambridge, Mass.

[Chaudhari et al., 1983] Chaudhari, R., Pham, S., and Garcia, O. N. (1983). Solution of an open problem on probabilistic grammars. *IEEE Transactions on Computers*, C-32(8):748–750.

[Chelba et al., 1997] Chelba, C., Engle, D., Jelinek, F., Jimenez, V., Khudanpur, S., Mangu, L., Printz, H., Ristad, E., Stolcke, A., Rosenfeld, R., and Wu, D. (1997). Structure and performance of a dependency language model. In *Proc. of Eurospeech 97*, volume 5, pages 2775–2778.

[Chelba and Jelinek, 1998] Chelba, C. and Jelinek, F. (1998). Exploiting syntactic structure for language modeling. In *Proc of COLING-ACL '98*, pages 225–231, Montreal.

[Chen et al., 1999] Chen, J., Bangalore, S., and Vijay-Shanker, K. (1999). New models for improving supertag disambiguation. In *Proceedings of the 9th Conference of the European Chapter of the Association for Computational Linguistics*, Bergen, Norway.

[Collins and Singer, 1999] Collins, M. and Singer, Y. (1999). Unsupervised Models for Named Entity Classification. In *In Proc. of WVLC/EMNLP-99*, pages 100–110.

[Cutting et al., 1992] Cutting, D., Kupiec, J., Pedersen, J., and Sibun, P. (1992). A practical part-of-speech tagger. In *Proc. of 3rd ANLP Conf.*, Trento, Italy. ACL.

[Dunning, 1993] Dunning, T. (1993). Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74.

[Elworthy, 1994] Elworthy, D. (1994). Does baum-welch re-estimation help taggers? In *Proceedings of 4th ACL Conf on ANLP*, pages 53–58, Stuttgart.

[Ersan and Charniak, 1996] Ersan, M. and Charniak, E. (1996). A statistical syntactic disambiguation program and what it learns. In Wermter, S., Riloff, E., and Scheler, G., editors, *Connectionist, Statistical and Symbolic Approaches in Learning for Natural Language Processing*, volume 1040 of *Lecture Notes in Artifical Intelligence*, pages 146–159. Springer-Verlag, Berlin.

[Fong and Wu, 1996] Fong, E. W. and Wu, D. (1996). Learning restricted probabilistic link grammars. In Wermter, S., Riloff, E., and Scheler, G., editors, *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing*, pages 173–187. Springer-Verlag.

[Goldman and Zhou, 2000] Goldman, S. and Zhou, Y. (2000). Enhancing supervised learning with unlabeled data. In *Proceedings of ICML'2000*, Stanford University.

[Hajič, 1998] Hajič, J. (1998). Building a syntactically annotated corpus: The prague dependency treebank. In *Issues of Valency and Meaning*, pages 106–132. Karolinum, Praha.

[Hajič and Hladká, 1998] Hajič, J. and Hladká, B. (1998). Tagging inflective languages: Prediction of morphological categories for a rich, structured tagset. In *Proceedings of COLING-ACL 98*, Université de Montréal, Montréal, pages 483–490.

[Harris, 1963] Harris, T. E. (1963). *The Theory of Branching Processes.* Springer-Verlag, Berlin.

[Horn and Johnson, 1985] Horn, R. A. and Johnson, C. R. (1985). *Matrix Analysis.* Cambridge University Press, Cambridge.

[Jelinek and Lafferty, 1991] Jelinek, F. and Lafferty, J. (1991). Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–323.

[Joshi and Schabes, 1991] Joshi, A. and Schabes, Y. (1991). Tree adjoining grammars and lexicalized grammars. In Nivat, M. and Podelski, A., editors, *Tree automata and languages.* North-Holland.

[Joshi, 1985] Joshi, A. K. (1985). Tree Adjoining Grammars: How much context Sensitivity is required to provide a reasonable structural description. In Dowty, D., Karttunen, I., and Zwicky, A., editors, *Natural Language Parsing*, pages 206–250. Cambridge University Press, Cambridge, U.K.

[Joshi, 1988] Joshi, A. K. (1988). An introduction to tree adjoining grammars. In Manaster-Ramer, A., editor, *Mathematics of Language.* John Benjamins, Amsterdam.

[Joshi et al., 1975] Joshi, A. K., Levy, L., and Takahashi, M. (1975). Tree Adjunct Grammars. *Journal of Computer and System Sciences.*

[Joshi and Schabes, 1992] Joshi, A. K. and Schabes, Y. (1992). Tree-adjoining grammar and lexicalized grammars. In Nivat, M. and Podelski, A., editors, *Tree automata and languages*, pages 409–431. Elsevier Science.

[Kaplan and Bresnan, 1983] Kaplan, R. and Bresnan, J. (1983). Lexical-functional Grammar: A Formal System for Grammatical Representation. In Bresnan, J., editor, *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, Massachusetts.

[Kay, 1989] Kay, M. (1989). Head driven parsing. In *Proc. of IWPT '89*, pages 52–62, Pittsburgh, PA.

[Kroch and Joshi, 1987] Kroch, A. S. and Joshi, A. K. (1987). Analyzing Extraposition in a Tree Adjoining Grammar. In Huck, G. and Ojeda, A., editors, *Discontinuous Constituents, Syntax and Semantics*, volume 20. Academic Press.

[Lafferty et al., 1992] Lafferty, J., Sleator, D., and Temperley, D. (1992). Grammatical trigrams: A probabilistic model of link grammar. In *Proc. of the AAAI Conf. on Probabilistic Approaches to Natural Language*.

[Lang, 1988] Lang, B. (1988). Parsing incomplete sentences. In *Proc. of the 12th International Conference on Computational Linguistics*, volume 1, pages 365–371, Budapest.

[Lang, 1994] Lang, B. (1994). Recognition can be harder than parsing. *Computational Intelligence*, 10(4).

[Lapata, 1999] Lapata, M. (1999). Acquiring lexical generalizations from corpora: A case study for diathesis alternations. In *Proceedings of 37th Meeting of ACL*, pages 397–404.

[Lapata and Brew, 1999] Lapata, M. and Brew, C. (1999). Using subcategorization to resolve verb class ambiguity. In Fung, P. and Zhou, J., editors, *Proceedings of WVLC/EMNLP*, pages 266–274.

[Lari and Young, 1990] Lari, K. and Young, S. J. (1990). The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 4:35–56.

[Lavelli and Satta, 1991] Lavelli, A. and Satta, G. (1991). Bidirectional parsing of Lexicalized Tree Adjoining Grammars. In *Proc. 5th EACL*, Berlin, Germany.

[Manning, 1993] Manning, C. D. (1993). Automatic acquisition of a large subcategorization dictionary from corpora. In *Proceedings of the 31st Meeting of the ACL*, pages 235–242, Columbus, Ohio.

[Marcken, 1995] Marcken, C. d. (1995). Lexical heads, phrase structure and the induction of grammar. In Yarowsky, D. and Church, K., editors, *Proceedings of the Third Workshop of Very Large Corpora*, pages 14–26, MIT, Cambridge, MA.

[Marcus et al., 1993] Marcus, M., Santorini, B., and Marcinkiewiecz, M. (1993). Building a large annotated corpus of english. *Computational Linguistics*, 19(2):313–330.

[Pietra et al., 1994] Pietra, S. D., Pietra, V. D., Gillett, J., Lafferty, J., Printz, H., and Ureš, L. (1994). Inference and estimation of a long-range trigram model. In Carrasco, R. and Oncina, J., editors, *Proc. of ICGI-94*. Springer-Verlag.

[Rambow and Joshi, 1995] Rambow, O. and Joshi, A. (1995). A formal look at dependency grammars and phrase-structure grammars, with special consideration of word-order phenomena. In Wanner, L., editor, *Current Issues in Meaning-Text Theory*. Pinter, London.

[Ratnaparkhi, 1996] Ratnaparkhi, A. (1996). A Maximum Entropy Part-Of-Speech Tagger. In *Proc. of the Empirical Methods in Natural Language Processing Conference*, University of Pennsylvania.

[Resnik, 1992] Resnik, P. (1992). Probabilistic tree-adjoining grammars as a framework for statistical natural language processing. In *Proc. of COLING '92*, volume 2, pages 418–424, Nantes, France.

[Sánchez and Benedí, 1997] Sánchez, J.-A. and Benedí, J.-M. (1997). Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):1052–1055.

[Schabes, 1992] Schabes, Y. (1992). Stochastic lexicalized tree-adjoining grammars. In *Proc. of COLING '92*, volume 2, pages 426–432, Nantes, France.

[Schabes et al., 1988] Schabes, Y., Abeillé, A., and Joshi, A. K. (1988). Parsing strategies with 'lexicalized' grammars: Application to Tree Adjoining Grammars. In *Proceedings of the* 12^{th} *International Conference on Computational Linguistics (COLING'88)*, Budapest, Hungary.

[Sikkel, 1997] Sikkel, K. (1997). *Parsing Schemata.* EATCS Series. Springer-Verlag.

[Soule, 1974] Soule, S. (1974). Entropies of probabilistic grammars. *Inf. Control*, 25:55–74.

[Srinivas, 1996] Srinivas, B. (1996). "Almost Parsing" technique for language modeling. In *Proc. ICSLP '96*, volume 3, pages 1173–1176, Philadelphia, PA.

[Srinivas, 1997a] Srinivas, B. (1997a). *Complexity of Lexical Descriptions and its Relevance to Partial Parsing.* PhD thesis, Department of Computer and Information Sciences, University of Pennsylvania.

[Srinivas, 1997b] Srinivas, B. (1997b). Performance Evaluation of Supertagging for Partial Parsing. In *Proceedings of Fifth International Workshop on Parsing Technology*, Boston, USA.

[Stevenson and Merlo, 1999] Stevenson, S. and Merlo, P. (1999). Automatic verb classification using distributions of grammatical features. In *Proceedings of EACL '99*, pages 45–52, Bergen, Norway.

[Stevenson et al., 1999] Stevenson, S., Merlo, P., Kariaeva, N., and Whitehouse, K. (1999). Supervised learning of lexical semantic classes using frequency distributions. In *SIGLEX-99.*

[Stolcke, 1995] Stolcke, A. (1995). An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201.

[Ushioda et al., 1993] Ushioda, A., Evans, D. A., Gibson, T., and Waibel, A. (1993). The automatic acquisition of frequencies of verb subcategorization frames from tagged corpora. In Boguraev, B. and Pustejovsky, J., editors, *Proceedings of the Workshop on Acquisition of Lexical Knowledge from Text*, pages 95–106, Columbus, OH.

[van Noord, 1994] van Noord, G. (1994). Head-corner parsing for TAG. *Computational Intelligence*, 10(4).

[Vijay-Shanker, 1987] Vijay-Shanker, K. (1987). *A Study of Tree Adjoining Grammars.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania.

[Vijay-Shanker and Weir, 1993] Vijay-Shanker, K. and Weir, D. J. (1993). The use of shared forests in TAG parsing. In *Proc of 6th Meeting of the EACL*, pages 384–393, Utrecht, The Netherlands.

[Webster and Marcus, 1989] Webster, M. and Marcus, M. (1989). Automatic acquisition of the lexical frames of verbs from sentence frames. In *Proceedings of the 27th Meeting of the ACL*, pages 177–184.

[Wetherell, 1980] Wetherell, C. S. (1980). Probabilistic languages: A review and some open questions. *Computing Surveys*, 12(4):361–379.

[Wright and Wrigley, 1989] Wright, J. H. and Wrigley, E. N. (1989). Probabilistic LR parsing for speech recognition. In *IWPT '89*, pages 105–114.

[Xia, 1999] Xia, F. (1999). Extracting tree adjoining grammars from bracketed corpora. In *Proceedings of 5th Natural Language Processing Pacific Rim Symposium (NLPRS-99)*, Beijing, China.

[Yarowsky, 1995] Yarowsky, D. (1995). Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. In *In Proc. 33rd Meeting of the ACL*, pages 189–196, Cambridge, MA.