COMBINING LABELED AND UNLABELED DATA IN STATISTICAL NATURAL LANGUAGE PARSING

Anoop Sarkar

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2002

Professor Aravind Joshi
Supervisor of Dissertation

Val Tannen
Graduate Group Chair

COPYRIGHT

Anoop Sarkar

2002

Acknowledgements

Prof. Aravind Joshi, my dissertation advisor has been my guide and mentor for the entire time that I spent at Penn. I thank him for all his academic help and personal kindness. The external member on my dissertation committee was Steven Abney, whose suggestions and advice have made the ideas presented here stronger. My dissertation committee members from Penn: Mitch Marcus, Mark Liberman and Martha Palmer provided questions whose answers shaped my dissertation proposal into the finished form in front of you. Many thanks to my academic collaborators; the work on prefix probabilities was done with Mark-Jan Nederhof and Giorgio Satta when they visited IRCS in 1998, the work on subcategorization frame learning was done in collaboration with Daniel Zeman when he visited IRCS in 2000. Thanks to B. Srinivas whose previous work provided the path to the experimental work in this dissertation. Thanks also to Paola Merlo and Suzanne Stevenson for discussions on their work on verb alternation classes. I also acknowledge the help of Woottiporn Tripasai in the extension of their work presented in this dissertation. Thanks to Mike Collins, Adwait Ratnaparkhi, Mark Dras, David Chiang, Dan Bikel, Tom Morton and Dan Gildea for helpful discussions. Fernando Pereira helped invaluably with his insightful suggestions. Moses Kimanzi and William Schuler worked with me in implementing parts of the XTAG parser which is documented within these pages. Past and current office-mates: Christy Doran, Fei Xia and Carlos Prolo were helpful, patient and accomodating. The linguists: John Bell, Tonia Bleam, David Embick, Alan Lee, Rashmi Prasad and Alexander Williams were personal and professional friends. Finally, without the help and support of Chung-hye Han and the patience of my parents, Leela and Dipesh Sarkar this thesis would remain unwritten.

Abstract

COMBINING LABELED AND UNLABELED DATA IN

STATISTICAL NATURAL LANGUAGE PARSING

Anoop Sarkar

Supervisor: Professor Aravind Joshi

Ambiguity resolution in the parsing of natural language requires a vast repository of knowledge to guide disambiguation. An effective approach to this problem is to use machine learning algorithms to acquire the needed knowledge and to extract generalizations about disambiguation decisions. Such parsing methods require a corpus-based approach with a collection of correct parses compiled by human experts. Current statistical parsing models suffer from sparse data problems, and experiments have indicated that more labeled data will improve performance. In this dissertation, we explore methods that attempt to combine human supervision with machine learning algorithms to try and extend accuracy beyond what is possible with the use of limited amounts of labeled data. In each case we do this by exposing a machine learning algorithm to unlabeled data in addition to the existing labeled data.

Most recent research in parsing has shown the advantage of having a lexicalized model, where the word relationships mediate knowledge about disambiguation decisions. We use Lexicalized Tree Adjoining Grammars (TAGs) as the basis of our machine learning algorithm since they arise naturally from the lexicalization of Context Free Grammars (CFGs). We show in this dissertation that probability measures applied to TAGs retain the simplicity of probabilistic CFGs along with its elegant formal properties and that while PCFGs need additional independence assumptions to be useful in statistical parsing, no such changes need to be made to probabilistic TAGs.

The main results presented in this dissertation are:

- We extend the Co-Training algorithm (Yarowsky 1995; Blum and Mitchell 1998), a machine learning technique for combining labeled and unlabeled data previously used with classifiers with 2/3 labels to the more complex problem of statistical parsing. Using empirical results based on parsing the Wall Street Journal corpus we show that training a statistical parser on the combined labeled and unlabeled data strongly outperforms training only on the labeled data.

- We present a machine learning algorithm that can be used to discover previously unknown subcategorization frames. The algorithm can then be used to label dependents of a verb in a treebank as either arguments or adjuncts. We use this algorithm to augment the Czech Dependency Treebank with argument/adjunct information.

- We extend a supervised classifier for automatically identifying verb alternation classes for a set of verbs so that it can be used on minimally annotated data. Previous work (Merlo and Stevenson 2001) provided a classifier for this task that used automatically parsed text. With the use of learning of subcategorization frames we construct the same type of classifier which now requires text annotated with part-of-speech tags and phrasal chunks.

In each of these results we use some existing linguistic resource that has been annotated by humans and add some further significant linguistic annotation by applying statistical machine learning algorithms.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Ambiguity resolution in the parsing of natural language requires a vast repository of knowledge to guide disambiguation. An effective approach to this problem is to use machine learning algorithms to acquire this knowledge and extract generalizations about disambiguation decisions. Such parsing methods require a corpus-based approach with a collection of correct parses compiled by human experts. Current statistical parsing models suffer from sparse data problems, and experiments have indicated that more labeled data will improve performance. In this dissertation, we explore methods that attempt to combine human supervision with machine learning algorithms that try and extend accuracy beyond what is possible with the use of limited amounts of labeled data. In each case we do this by exposure to unlabeled data in addition to the existing labeled data.

Most recent research in parsing has shown the advantage of having a lexicalized model, where the word relationships mediate knowledge about disambiguation decisions. We use Lexicalized Tree Adjoining Grammars (TAGs) as the basis of our machine learning algorithm since they arise naturally from the lexicalization of Context Free Grammars (CFGs). We show that statistical models of parsing with TAGs provide a natural boundary between statistical models that build clausal structure and full-blown statistical parsing which combines clauses to form sentence structures. We show in this dissertation that probability measures applied to TAGs retain the simplicity of probabilistic CFGs along with its elegant formal properties.

1.1 Outline of the Dissertation

This dissertation presents several results that are related to statistical parsing. In each of these results we use some existing linguistic resource that has been annotated by humans and added some further significant linguistic annotation by applying statistical machine learning algorithms. The various results that relate to this overall theme are organized into the following parts in this dissertation.

- Chapter 3 presents a novel Co-Training method for statistical parsing. Using empirical results based on parsing the Wall Street Journal corpus we show that training a statistical parser on the combined labeled and unlabeled data strongly outperforms training only on the labeled data.

- Chapter 4 presents a machine learning algorithm that can be used to discover previously unknown subcategorization frames. The algorithm can then be used to label dependents of a verb in a treebank as either arguments or adjuncts.

- Chapter 5 presents a classifier for automatically identifying verb alternation classes for a set of verbs which extracts features from distributional information about the local clausal context of verbs exploiting very minimal annotation of the data such as part-of-speech tags and phrasal chunks.

The rest of this chapter provides an introduction to the problem of statistical parsing and gives a historical overview of the methods and important concepts used in this field. Chapter 2 provides some motivation for the use of Tree Adjoining Grammars (TAGs) for statistical parsing and introduces some basic results in the field of probabilistic TAGs. This chapter also shows that probabilistic TAGs retain the simplicity of probabilistic CFGs along with its elegant formal properties and while PCFGs need additional independence assumptions to be useful in statistical parsing, no such changes need to be made to probabilistic TAGs.

In Chapter 6 we provide the parsing algorithm used in the experiments that use a TAG-based statistical parser in this dissertation. We also include various performance evaluations of the parser in this chapter.

Appendix A gives a formal justification for the algorithm given in Chapter 2 showing whether a given stochastic TAG is consistent by showing a reduction of the TAG derivation process to a multitype Galton-Watson branching process [Har63].

Appendix B gives a full treatment of the computation of prefix probabilities introduced in Chapter 2. It uses a program transformation approach to translate the inside probability computation into a prefix probability computation for stochastic TAGs.

1.2 Introduction to Statistical Parsing

Parsing and the general issue of syntactic analysis is part of the general scientific analysis within the fields of artificial intelligence and cognitive science of the human ability to map sounds into complex (recursive) semantic structures. Syntactic processing also is relevant for issues in human-computer interaction (through both speech and text). Also, breakthroughs in parsing are likely to benefit various engineering problems such as machine translation, information extraction, summarization, among others.

Statistical parsing tries to solve one problem among many others in the overall field of syntactic analysis. This problem is that of ambiguity. Ambiguity resolution in the parsing of natural language requires a vast repository of knowledge to guide disambiguation. An effective approach to this problem is to use machine learning algorithms to acquire this knowledge and extract generalizations about disambiguation decisions. Such parsing methods require a corpus-based approach with a limited collection of correct parses compiled by human experts.

1.2.1 Types of Syntactic Analysis

There are three major types of syntactic analyses that are produced by current parsers:

- Constituent structures
- Dependency analysis
- Non-recursive analysis into text chunks

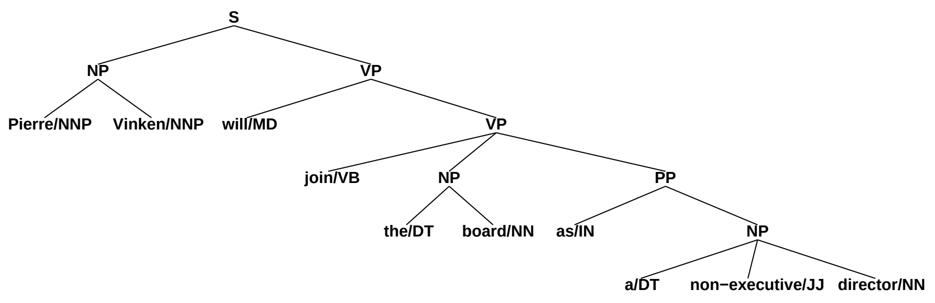


Figure 1.1: A sample constituent analysis of a sentence

Constituent Structures

Constituent structures for language exploit the use of abstract labels for their description. They typically also employ various complex relationships over these abstract labels such as the annotation of the dislocation of various constituents in various natural language phenomena like relativization, ellipsis, right-node raising, among others.

Examples of such a kind of analysis in building Treebanks is the Penn English Treebank, the Penn-Helsinki Middle-English Treebank, the Penn Korean Treebank and the German Negra Treebank.

An example of constituent analysis is given in Figure 1.1. Evaluation of parsers that are trained on this kind of data is usually done on the brackets recovered by the parser.

Complex structural labels and descriptions in the representation that includes empty elements means that such parses are hard to induce using purely self-organizing methods.

The recovery of recursive constituent structures is at least polynomial in the length of the input. Techniques vary from $O(n^3)$ to $O(n^6)$ for fully bottom-up techniques. Left to right shift reduce algorithms can be exploited to get observable linear time performance.

Dependency Analysis

Dependency analysis provides word to word relationships that are often typed by the assignment of a label. Typically dependency analyses shun the use of abstract relationships between phrases

0	Pierre/NNP	1
1	Vinken/NNP	3
2	will/MD	3
3	join/VB	TOP
4	the/DT	5
5	board/NN	3
6	as/IN	3
7	a/DT	9
8	non-executive/JJ	9
9	director/NN	6

Figure 1.2: A sample dependency analysis of a sentence

relying only on word relationships. As a result, different corpora annotated with dependency structure typically give very different (and usually ad-hoc) treatments of crossing dependencies, and phenomena that target constituents (e.g: relativization and ellipsis).

Examples of such dependency structure corpora include the Czech Prague Treebank, Japanese EDR corpus and the Hindi Treebank from LTRC, Hyderabad.

An example of dependency analysis is given in Figure 1.2. Most dependency annotations also add some more information such as a label indicating linguistic information (such as subject, object, etc.) about the type of link between two words. Evaluation of parsers that are trained on this kind of data is usually done on accuracy of the dependency links produced by the parser.

It is important to note that it is possible to map between dependency analyses and constituent structures. This is has been done when parsing the Czech Treebank (from dependencies to constituent structure [CHRT99]). And also when parsing the Penn Treebank (from constituent structure to word-word dependencies [Eis96].

While the structures are simpler in dependency analysis, the worst time complexity analysis for dependency parsing is at least as bad as that for constituent analysis.

Chunking and Shallow Parsing

A third type of syntactic analysis is to find non-recursive or basal phrases in text which is often useful in many applications where a full syntactic analysis is unnecessary. Since only non-recursive constituent analysis is done, typically the complexity of this task in linear in the length of the input.

```
>> [ John/NNP ] saw/VBD [the/DT cat/NN]
        [the/DT dog/NN] liked/VBD ./.
>> [ John/NNP Smith/NNP ] ,/, [ president/NN ]
        of/IN [ IBM/NNP ] ./.
>> [ Pundits/NNS ] condemned/VBD [ terrorism/NN ]
        and [ assassination/NN ] ./.
```

Figure 1.3: Chunking analysis of a sentence

```
[nx
            nns             techniques
    [pp
            in              for
        [nx
            vbg             bootstrapping
            jj              broad
            nn              coverage
            nns             parsers
        ]
    ]
]
```

Figure 1.4: Cascades of chunks in a sentence

Figure 1.3 shows the basal noun phrases that can be recovered by a chunk parser. Cascades of chunkers can be used to extend such shallow parsing techniques towards full syntactic analysis (see for example a sample output from Steve Abney's chunker [Abn97] in Figure 1.4). For more on chunking see [Abn97]. These techniques typically have problems recovering sentential phrase boundaries accurately when they rely on exclusively finite-state methods.

1.2.2 Parsing Algorithms

A generative linguistic theory is implemented in a formal system to produce the set of grammatical strings and rule out ungrammatical strings. In a probabilistic setting, this system produces a probability distribution over the strings in the language, assigning probability of zero to strings not in the language. Such a formal system has an associated decision problem: given a string, can it be *recognized* by the formal system. An algorithm for this decision problem is called a recognizer.

- Consider the CFG G:

    1. $S \rightarrow S\ S$
    2. $S \rightarrow a$

  $L(G) = a^i$ for $i >= 1$

- The recognition question: does the string *aaa* belong to $L(G)$?

  - Input: *aaa*
  - Output: {*yes*, *no*}

Figure 1.5: The recognition problem for a grammar formalism

If a string is generated by a grammar, the record of the steps taken to recognize the string is called a *parse* derived by a parsing algorithm.

Recognition and parsing algorithms should be decidable. Preferably the algorithm should be polynomial since this enables computational implementations of linguistic theories. Elegant, polynomial-time algorithms exist for formalisms like context-free grammars and tree-adjoining grammars. See for example a simple parser for probabilistic CFGs shown in Figure 1.7 which takes as input a PCFG of the kind shown in Figure 1.2.2. In this algorithm the function `extractbest` traverses the back pointers stored in the `B` array to find the best derivation for an input string. A simple analysis of the algorithm shows that the worst case complexity of the parser for any input string is $\mathcal{O}(G^2 n^3)$ where n is the length of the input and G is a constant depending on the grammar. Since the grammar is considered invariant, the G^2 term is usually ignored. However, if each non-terminal also refers to some terminal symbol (the so-called bi-lexical CFGs) in the input then the complexity becomes $\mathcal{O}(n^5)$. [ES99] has shown that this worst case complexity for bi-lexical grammars can be reduced back to $\mathcal{O}(n^3)$ with some grammar constant.

1.2.3 Ambiguity in Natural Language Grammars

Statistical parsing deals with the problem of ambiguity which causes the number of derivations to grow exponentially. For example, for increasing lengths of the input string the number of derivations obtained by the grammar is shown in Figure 1.2.3.

```
$grammar{"S S"} = ["S"];
$grammar{"a"} = ["S"];
@start = ("S");
$prob{"S -> S S"} = 0.3;
$prob{"S -> a"} = 0.7;

@input = (a, a, a, a, a);
```

Figure 1.6: Example probabilistic CFG grammar as input for parser in Figure 1.7

The issue of syntactic ambiguity in natural language is brought into sharp focus when you consider the algebraic character of parses produced by a CFG. Using the description of CFG derivations in terms of power series the paper by [CP82] showed how even simple natural language grammars can be highly ambiguous. See [Sal73] for further background to the use of power series to describe derivations of context-free grammars.

Let us take, for example, the power series for the grammar: `NP → cabbages | kings | NP and NP`:

```
NP = cabbages + cabbages and kings
+ 2 (cabbages and cabbages and kings)
+ 5 (cabbages and kings and cabbages and kings)
+ 14 ...
```

The coefficients equal the number of parses for each NP string. These ambiguity coefficients are Catalan numbers:

$$Cat(n) = \begin{pmatrix} 2n \\ n \end{pmatrix} - \begin{pmatrix} 2n \\ n-1 \end{pmatrix}$$

$\begin{pmatrix} a \\ b \end{pmatrix}$ is the *binomial coefficient*.

$$\begin{pmatrix} a \\ b \end{pmatrix} = \frac{a!}{(b!(a-b)!)}$$

$Cat(n)$ also provides exactly the number of parses for sentences that contain multiple prepositional phrases with multiple possibilities of attachment, such as:

```
$N = $#input + 1;
@A = ();  # len^2
%B = ();  # backpointers: len^2 * NTs
%P = ();  # prob: len^2 * NTs

$HUGE_VAL = 1048576; $DEFAULT  = -$HUGE_VAL;

for ($i = 0; $i < $N; $i++) {
    # nt -> input[i]
    foreach $nt (@{$grammar{"$input[$i]"}}) {
        $j = $i+1;
        if (!(defined $B{$i}{$j}{$nt})) {
            push @{$A[$i][$j]}, $nt;
            print STDERR "inserting A[$i][$j] = $nt\n";
        }
        $np = log($prob{"$nt -> $input[$i]"});
        $op = (defined $P{$i}{$j}{$nt}) ? $P{$i}{$j}{$nt} : $DEFAULT;
        if ($np >= $op) {
            $P{$i}{$j}{$nt} = $np;
            $B{$i}{$j}{$nt} = "/$input[$i]/";
        }}}

for ($j = 2; $j <= $N; $j++) {
    for ($i = $j-2; $i >= 0; $i--) {
        for ($k = $i+1; $k < $j; $k++) {
            foreach $nt1 (@{$A[$i][$k]}) {
                foreach $nt2 (@{$A[$k][$j]}) {
                    # nt -> nt1 nt2
                    foreach $nt (@{$grammar{"$nt1 $nt2"}}) {
                        if (!(defined $B{$i}{$j}{$nt}))
                            push @{$A[$i][$j]}, $nt;
                            print STDERR "inserting A[$i][$j] = $nt\n";
                        }
                        $np = $P{$i}{$k}{$nt1} + $P{$k}{$j}{$nt2} +
                            log($prob{"$nt -> $nt1 $nt2"});
                        $op = (defined $P{$i}{$j}{$nt}) ? $P{$i}{$j}{$nt} : $DEFAULT;
                        if ($np >= $op) {
                            $P{$i}{$j}{$nt} = $np;
                            $B{$i}{$j}{$nt} = "$k $nt1 $nt2";
                        }}}}}}}

$bp = $DEFAULT;
$best = '';
foreach $s (@start) {
    my $rx = extractbest(0, $N, $s);
    next if ($rx eq '');
    $np = $P{0}{$N}{$s};
    if ($np >= $bp) { $best = $rx; $bp = $np; }
}
print "$best\n";
```

Figure 1.7: Simple Perl implementation of a parser for Probabilistic CFGs which reports a single best parse given an input string

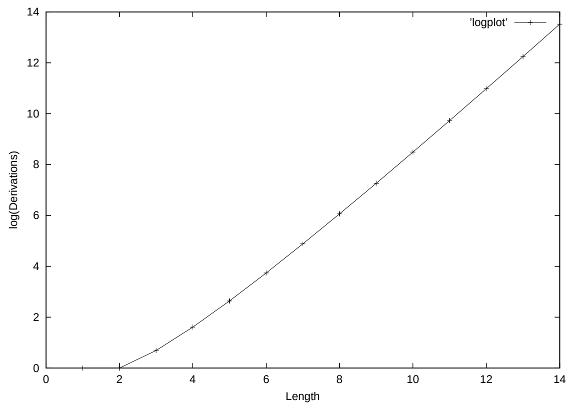


Figure 1.8: Number of derivations for strings in `L(G) = a a ...` for the grammar `G = S` $\rightarrow$ `S S`

```
John saw the man
   on the hill
   with the telescope
```

The various attachments describe the five different meanings possible in this sentence.

Other sub-grammars that can be written in a natural language grammar are simpler:

$$ADJP \rightarrow adj\ ADJP \mid \epsilon$$

$$ADJP = 1 + adj + adj^2 + adj^3 + \dots$$

$$ADJP = \frac{1}{1-adj}$$

Since in a natural language grammar these individual constructions have to be combined. So we also have to consider power series of combinations of sub-grammars. For example the combination of `NP` and `VP` grammars is given by the rule: `S = NP · VP` . This gives analyses for strings such as:

```
( The number of products over sales ... )
( is near the number of sales ... )
```

As can be seen in this example, both the `NP` subgrammar and the `VP` subgrammar power series have Catalan coefficients. The power series for the `S` $\rightarrow$ `NP VP` grammar is the multiplication:

$$(N \sum_i Cat_i \, (P \, N)^i) \cdot (is \sum_j Cat_j (P \, N)^j)$$

In a parser for this grammar, this leads to a cross-product:

$$L \times R = \{ (l, \; r) \mid l \in L \,\&\, r \in R \}$$

These examples show in concrete terms that ambiguity is pervasive in any grammar that aims to capture natural language sentences. Rather than produce massively ambiguous outputs when analysing naturally occuring sentences we wish to find a method for ambiguity resolution that will produce a single best parse (or at least a ranked list of parses). We shall see how statistical parsing aims to solve this problem.

Ambiguity Resolution: Prepositional Phrases in English

One method to deal with the ambiguity prevalent in prepositional phrase grammars is to use a supervised learning method which learns proper attachments. We can use the words such as for example, the preposition used, as indicative features to disambiguate the attachment. As we shall see, this provides a simple method to deal with the ambiguity in prepositional phrase grammars.

In order to have an effective supervised learning method we need an adequate amount of data labeled with the right answer. The supervised learner will learn from this annotated data how to make decisions on unseen data. Figure 1.9 shows an example of a list of attachment decisions given for a set of simple PP attachment decisions (only disambiguating between a single noun and verb phrase attachment) which were extracted from the Penn Treebank [RRR94].

The difficulty of the task can be observed in the table shown in Figure 1.2.3. The average human refers to the average performance of a team of annotators on this task. Inter-annotator disagreement can be seen in the figure of 93.2% even when the full sentence is available. This disagreement increases noise in the data and affects performance of supervised learning methods including supervised statistical parsing[1].

There have been several methods which have been proposed for the supervised learning of

[1]Perhaps methods such as those given in [ASS99] can be used to deal with this kind of problems

V	N1	P	N2	Attachment
join	board	as	director	V
is	chairman	of	N.V.	N
using	crocidolite	in	filters	V
bring	attention	to	problem	V
is	asbestos	in	products	N
making	paper	for	filters	N
including	three	with	cancer	N

Figure 1.9: Annotated data providing prepositional phrase attachments

Method	Accuracy
Always noun attachment	59.0
Most likely for each preposition	72.2
Average Human (4 head words only)	88.2
Average Human (whole sentence)	93.2

Figure 1.10: Baseline models and annotator agreement on the PP attachment task

PP attachment. Let us take one example of a supervised learning method for learning prepositional phrase attachment as explored in [CB95]. The method involves the learning of a probability model $p(d \mid v, n1, p, n2)$ where d is a binary decision $\{0, 1\}$ between verb and noun attachment and $v, n1, p, n2$ are the head-words in each position involved in a attachment decision. So if $p(1 \mid v, n1, p, n2) >= 0.5$ the learner chooses noun attachment. The main problem with this probability model is that of sparse data: the learner is unlikely to see each of the four words $v, n1, p, n2$ in unseen data. To deal with this issue, the model proposed in [CB95] for PP attachment adds several back-off levels ([Kat87]). The entire model is given in Figure 1.2.3.

The results obtained by this model are 84.1% accuracy (cf. the accuracy of the various baseline models and human performance in Figure 1.2.3). If we analyze why this particular algorithm works as well it does we encounter the following points in its favor (some of these were also used in previous models).

- Lexicalization helps disambiguation by capturing selectional preferences. This was also the insight behind early work in statistical parsing [BJL$^+$93, Mag95]

- Smoothing deals with sparse data and improves prediction. Further improvements can

$$
\begin{aligned}
p(1 \mid v, n1, p, n2) = \; & \lambda(c_1) && \cdot \; p(1 \mid c_1 = v, n1, p, n2) \\
+ \; & \lambda(c_2 + c_3 + c_4) && \cdot \; p(1 \mid c_2 = v, n1, p) \\
& && \cdot \; p(1 \mid c_3 = v, p, n2) \\
& && \cdot \; p(1 \mid c_4 = n1, p, n2) \\
+ \; & \lambda(c_5 + c_6 + c_7) && \cdot \; p(1 \mid c_5 = v, p) \\
& && \cdot \; p(1 \mid c_6 = n1, p) \\
& && \cdot \; p(1 \mid c_7 = p, n2) \\
+ \; & \lambda(c_8) && \cdot \; p(1 \mid c_8 = p) \\
+ \; & (1 - \textstyle\sum_i \lambda(c_i)) && \cdot \; 1.0 \;\; \text{(default is noun attachment)}
\end{aligned}
$$

Figure 1.11: The back-off probability model for PP attachment

be seen in PP attachment accuracy if we add information about word classes and senses, see [SN97].

- Uses the head of the phrase, in this case, the preposition as privileged.
- Even single counts are trusted instead of treating them as noise due to limited amount training data and the sparse nature of lexical information.

Similar insights about lexicalization and the priviliged status of the semantic head of a construction led to lexicalization of grammars in mathematical linguistics and all-paths parsing; cf. tree-adjoining grammars (TAG) and combinatory categorial grammars (CCG).

By dealing with the sparse data problem by bringing in external resources such as Wordnet to back off to word classes and doing some simple word-sense disambiguation as was pursued in [SN97] leads to improvement in performance on this task to 88%.

The question subsequently asked by many researchers was whether we can take this insight about PP attachment models and improve on parsing performance using Probabilistic CFGs in finding the best parse for a given sentence.

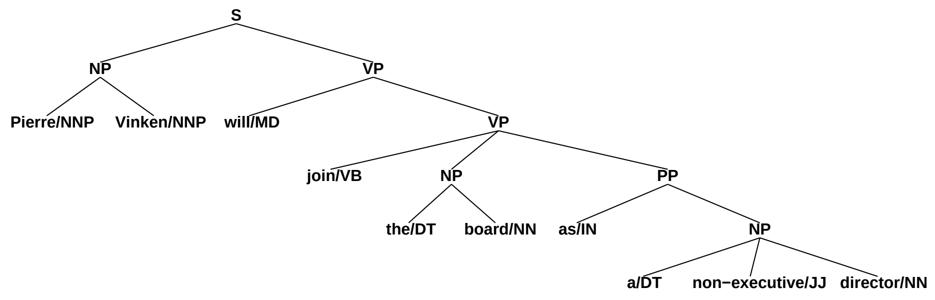


Figure 1.12: Annotated data for statistical parsing: an example sentence from the Penn WSJ Treebank.

Supervised Models for Parsing

Analogous to the PP attachment task, in order to build supervised models for parsing we need annotated data with which to train our parsers. Such annotated data is stored in the form of a Treebank where the correct analysis according to some linguistic theory is stored for each sentence in a corpus of text. The largest such collection to date is the Penn Treebank which is an annotated collection of a million words of Wall Street Journal text. An example sentence from the Penn Treebank with its annotation is shown in Figure 1.2.3.

In order to apply our experience with prepositional phrase attachment to this more complex problem of finding the best parse for a given sentence, we would like to exploit lexical information. Such information can be used to improve our conditioning context when building up a parse tree. A common method to add such information is the use of heuristic head-percolation rules (see Figure 1.13). We discuss the role of lexical information in statistical parsing in detail in Chapter 2.

Given annotated data of the type shown in Figure 1.2.3 we can frame parsing as a supervised learning task. The problem is reduced to to the induction of a function (see [Col97]):

$$f : \mathcal{S} \rightarrow \mathcal{T}$$

where $\mathcal{S}$ are sets of sentences and $\mathcal{T}(S)$ are sets of trees for a given sentence S. This function can be used to produce the best tree $T_i \in \mathcal{T}(S)$ given a sentence $S_i \in \mathcal{S}$. A statistical parser builds model $P(T, S)$ for each (T, S). The best parse is then:

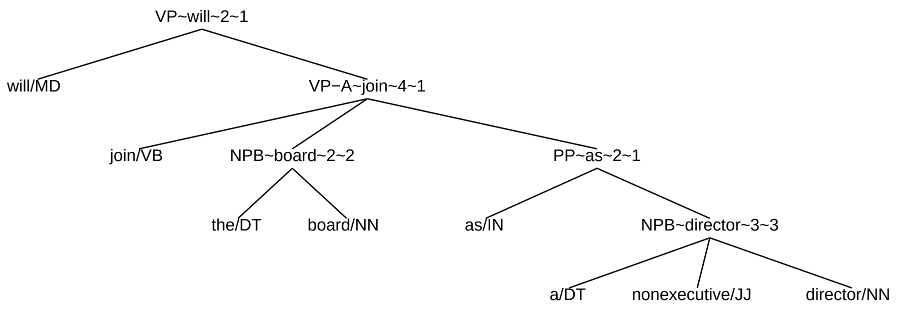


Figure 1.13: Lessons learned from prepositional phrase attachment applied to parsing [Mag95].

$$\arg\max_{T \in \mathcal{T}(S)} P(T, S)$$

A *history-based approach* to statistical parsing maps (T, S) into a decision sequence $d_1, \ldots, d_n$. The probability of tree T for sentence S is:

$$P(T, S) = \prod_{i=1 \ldots n} P(d_i \mid \phi(d_1, \ldots, d_{i-1}))$$

where, ϕ is a function that groups histories into equivalence classes.

PCFGs can be viewed as a history-based model using leftmost derivations. A tree with rules $\langle \gamma_i \rightarrow \beta_i \rangle$ is assigned a probability $\prod_{i=1}^{n} P(\beta_i \mid \gamma_i)$ for a derivation with n rule applications.

$$\begin{aligned}
T_{best} &= \arg\max_T P(T \mid S) \\
&= \arg\max_T \frac{P(T, S)}{P(S)} \\
&= \arg\max_T P(T, S) \\
&= \prod_{i=1 \ldots n} P(RHS_i \mid LHS_i)
\end{aligned}$$

$$\begin{aligned}
\text{Bracketing recall} &= \frac{\text{num of correct constituents}}{\text{num of constituents in the goldfile}} \\
\text{Bracketing precision} &= \frac{\text{num of correct constituents}}{\text{num of constituents in the parsed file}} \\
\text{Complete match} &= \text{\% of sents where recall \& precision are both 100\%} \\
\text{Average crossing} &= \frac{\text{num of constituents crossing a goldfile constituent}}{\text{num of sents}} \\
\text{No crossing} &= \text{\% of sents which have 0 crossing brackets} \\
\text{2 or less crossing} &= \text{\% of sents which have} \leq 2 \text{ crossing brackets}
\end{aligned}$$

Figure 1.14: Parseval metrics for parser evaluation

Evaluation of Statistical Parsers

An important requirement for the endeavour of statistical parsing, indeed for corpus based NLP in general, is to have a consensus on the evaluation metrics for a given problem and its associated corpus type.

For statistical parsing this evaluation method was set at a workshop meeting of researchers into parsing. The result was a group of measures with which to evaluate parsers that report labeled constituents in their output. This evaluation measure was called *Parseval*. A program called *EVALB* was written by Collins and Sekine that encodes these metrics. Parseval evaluation metrics are shown in Figure 1.2.3. The most commonly reported figures are the precision and recall of the labeled brackets or constituents reported by the parser.

An exhaustive literature survey of supervised statistical parsing methods and results on probabilistic grammars already appears in Chapter 2 of [Col98]. In the remainder of the chapter we shall look at some more recent research in the field of statistical parsing particularly directed towards the effective use of the existing amounts of labeled data available for training a parser.

1.2.4 Some Current Directions in Statistical Parsing

In this section we shall look at some current directions in statistical parsing. In particular we will look at the following aspects of statistical parsing:

System	$\leq 40wds$ LP	$\leq 40wds$ LR	$\leq 100wds$ LP	$\leq 100wds$ LR
(Magerman 95)	84.9	84.6	84.3	84.0
(Collins 99)	88.5	88.7	88.1	88.3
(Charniak 97)	87.5	87.4	86.7	86.6
(Ratnaparkhi 97)			86.3	87.5
(Charniak 99)	90.1	90.1	89.6	89.5
(Collins 00)	90.1	90.4	89.6	89.9
Voting (HB99)	92.09	89.18		

Figure 1.15: Statistical parsing results using Parseval for various lexicalized PCFG models. LP = labeled precision. LR = labeled recall

- Voting methods for parser combination
- Discriminative methods and parse re-ranking
- Combination of unlabeled data with labeled data

Voting methods for parser combination

A promising technique which has been pursued with great deal of success in speech recognition is the combination of different systems by voting between their predictions using various methods. [Hen98] and [HB00] explore various techniques in the combination of existing statistical parsers that have been trained and tested on the WSJ Penn Treebank.

There are two main methods that can be used to combine the output of different statistical parsers.

Parse Hybridization In this technique individual constituents are selected from each parser.

Parser Switching Learn which is the best parser for a given sentence by using features from the input sentence.

For both of these methods [Hen98] gives some machine learning methods which can be trained on some held-out data.

- Parse Hybridization

$$\arg\max_{\pi(c)} P(\pi(c) \mid M_1(c) \dots M_k(c)) =$$

$$\arg\max_{\pi(c)} \frac{P(M_1(c) \dots M_k(c) \mid \pi(c)) \cdot P(\pi(c))}{P(M_1(c) \dots M_k(c))}$$

$$\arg\max_{\pi(c)} P(\pi(c)) \cdot \prod_{i=1}^{k} \frac{P(M_i(c) \mid \pi(c))}{P(M_i(c))}$$

$$\arg\max_{\pi(c)} P(\pi(c)) \cdot \prod_{i=1}^{k} P(M_i(c) \mid \pi(c))$$

Figure 1.16: Naive Bayes classifier for parse hybridization

  - Constituent Voting
  - Naive Bayes

- Parser Switching
  - Similarity Switching
  - Naive Bayes

In parse hybridization, constituent voting is a simple vote among the different parsers involved in the experiment (in [Hen98] the experiments used three parsers). The constituent that gets the most votes wins with a priority given to the highest performing parser in case none of the parsers agree.

The naive bayes method also does voting but it learns a classifier to perform the voting instead of doing simple voting. Let $\pi(c) = 1$ when constituent c should be included in the output parse. $M_i(c) = 1$ indicates that parser i suggests that constituent c should be in the output parse. The model picks likely constituents ($P > 0.5$) to be in the output parse where P is chosen using a naive bayes classifier shown in Figure 1.2.4.

Parser switching is simpler than combining consitutent since only one parser is chosen for a given sentence. The rationale for preferring this method was that each parser is decoding the best parse according to some generative or conditional model and parse hybridization could combine together individual consituents but fail in combining them together to form the best parse. However,

Reference/System	LP	LR
Average Individual Parser	87.14	86.91
Best Individual Parser	88.73	88.54
Parser Switching Oracle	93.12	92.84
Maximum Precision Oracle	100.00	95.41
Similarity Switching	89.50	89.88
Constituent Voting	92.09	89.18

Figure 1.17: Performance of various parser combination methods

as shown by the results in Figure 1.2.4 from [Hen98], constituent voting (the simplest method) outperforms the other methods. This figure also shows (see the oracle precision/recall figures) that voting methods could be improved further.

Discriminative methods for parse reranking

Experiments in [Rat98] showed that if the parser was allowed upto 20 chances to get the best parse, accuracy could be as high as 94% avg LP/LR on the standard parsing task on Section 23. For most statistical parsers, 20 ranked guesses are easy to produce (as $T_{best} \ldots T_{best-19}$). This leads to the idea of automatic reranking which can be used to produce a more accurate parser. [Col00] showed that reranking can improve parsing accuracy.

The approach taken in [Col00] was as follows. Let $x_{i,j}$ be the jth parse of the ith sentence. The statistical parser produces estimates $L(x_{i,j}) = \log Q(x_{i,j})$ which is the log probability of a parse. The task of an automatic reranking program is to learn a ranking function $F(x_{i,j})$. A baseline ranking is given by just trusting the statistical parser: $F(x_{i,j}) = L(x_{i,j})$

Intuitively, a re-ranking program should exploit features that were unavailable to the original statistical parser either because it would have cause sparse data problems in a generative model or because decoding (parsing) using the model was done purely bottom-up. Let us provide parameters which will weight a new set of features used in re-ranking. These parameters are defined as $\alpha = \{\alpha_0, \ldots, \alpha_m\}$. The new re-ranking function is now given as:

$$F(x_{i,j}, \alpha) = \alpha_0 L(x_{i,j}) + \sum_{s=1}^{m} \alpha_s h_s(x_{i,j})$$

where h_s are features extracted from the parses produced by the statistical parser. For example,

$$h_s(x) = \begin{cases} 1 & \text{if } x \text{ contains rule } \texttt{S} \rightarrow \texttt{NP VP} \\ 0 & \text{otherwise} \end{cases}$$

Since we now have the n-best output of the parser on a sentence and we have access to the correct parse in that set, the objective of learning a re-ranking function can now be stated as the minimization of the *ranking error rate*. This error rate is defined as the number of times a lower scoring parse is ranked above best parse in the baseline ranking of the parser.

[Col00] gives various discriminative methods to minimize the ranking error rate. Various non-local features can now be used. These features were unavailable within a top-down generative model. Figure 1.2.4 give the set of features used in the re-ranking experiment described in [Col00]. Using this method, [Col00] showed that parsing performance improved with a 13% decrease in the labeled constituent error rate: LP 90.4% LR 90.1% ($\leq$ 40 wds) and LP 89.6% LR 89.9% ($\leq$ 100 wds).

Limited amounts of labeled data

In this section we review some methods which attempt to deal with the problem of limited amounts of labeled data.

The Inside-Outside Algorithm The Inside-Outside Algorithm is a self-organizing method that finds parameter values for probabilistic CFGs. Expected usage of each production in the CFG is treated as hidden data and discovered using EM-based re-estimation using the Maximum Likelihood principle. The algorithm finds parameter values for PCFGs that reduce the training set entropy.

It does this by initializing the PCFG rule probabilities to random values. It then computes the following probabilities α and β , for each non-terminal X based on an input training (unlabeled) corpus. Then for each rule in which X is the lhs, values of α and β are used to compute new expected usage of that rule by using the maximum likelihood principle

- *inside probability*: $P(X \overset{*}{\Rightarrow} t_j \ldots t_k)$
  written as $\beta(X, j, k)$

Rules context-free rules, e.g. `VP` $\rightarrow$ `PP VBD NP NP SBAR`

Bigrams Pairs of non-terminals to the left and right of the head, e.g. `(Right, VP, NP, NP)`, `(Right, VP, NP, SBAR)`, `(Right, VP, PP, STOP)` and `(Left, VP, PP, STOP)`

Grandparent Rules Same as **Rules** including the parent of the LHS

Grandparent Bigrams Same as **Bigrams** including parent of the LHS

Lexical Bigrams Same as **Bigrams** but with lexical heads

Two-level Rules Same as **Rules** but with the LHS expanded to an entire rule

Two-level Bigrams Same as **Bigrams** but with the LHS expanded to an entire rule

Trigrams All trigrams within a rule, e.g. `(VP, STOP, PP, VBD!)`

Head-Modifiers All head-modifier pairs with grandparent non-terminals, e.g. `(Left, VP, VBD, PP)`

PPs Lexical trigrams for PPs, e.g. (NP (NP the president) (PP of (NP IBM))) produces `(NP, NP, PP, NP, president, of, IBM)` as well as `(NP, NP, PP, NP, of, IBM)`

Distance Head-Modifiers Distance between head words in a CFG attachment rule

Closed-class Lexicalization Add closed-class words to non-lexicalized non-terminals in above rules (except last 3)

Figure 1.18: Non-local features used in [Col00]

- *outside probability*: $P(S \overset{*}{\Rightarrow} t_1 \ldots t_{j-1}\ X\ t_{k+1} \ldots t_n)$
  written as $\alpha(X, j, k)$

This step is then iterated to convergence. The theorem on the EM family of algorithms by [DLR77] (of which the IO algorithm is a special case) states that likelihood is always non-decreasing. This technique is theoretically well motivated, but computationally expensive $O(n^3)$ per sentence in each iteration for PCFGs.

Even for small manageable datasets and small grammars, the IO algorithm leads to linguistically uninteresting grammars as discussed in [Mar95]. The main caution in using completely unsupervised methods such as the IO algorithm is that minimizing corpus likelihood might not lead to minimization of error rate (which is the real evaluation measure). More discussion of this is given in Chapter 3.

A more interesting use of outside probabilities is to find the most likely constituents in the analysis of an input sentence. [Goo96] explores this view and shows that one can do better at labeled constituent measures if the parser explicity maximizes constituent probabilities (using the outside probability) instead of finding the best parse tree. On the other hand, the standard method of finding the best tree does better when that is the measure being evaluated.

Active learning Active learning (and more narrowly the field of sample selection) aims to discover which data from a pool of unlabeled data when annotated by hand would be most informative for a machine learning algorithm. This could be restated as the question of selecting a subset from a large pool of text to be annotated to create a training set. Thus, active learning provides a better approach than blindly labeling an arbitrary set of data. However, the drawbacks of sample selection are that it is often biased to a particular learning model. A common answer to this problem is to use a committee of learners (see [ED96, CAL94])

We will discuss one experiment which used sample selection to select data for a statistical parser. In [Hwa01, Hwa00] new sentences were selected to be annotated based on their Training Utility Function which was defined to be some appropriate metric on the usefulness of that sentence when annotated to the future performance of a statistical parser. Given such an evaluation function f, Figure 1.2.4 shows an algorithm for sample selection which is an iterative method to improve parser performance using an expert human to label new data for the parser.

```
Algorithm:

$U$ is a set of unlabeled candidates
$L$ is a set of labeled training examples
$M$ is the current model
Select($n, U, M, f$):
returns $n$ candidates picked from $U$ based on an evaluation function $f$ and a model $M$

Initialize:
    $M$   $\leftarrow$   Train($L$)

Repeat:
    N     $\leftarrow$   Select($n, U, M, f$)
    $U$   $\leftarrow$   $U$ - N
    $L$   $\leftarrow$   $L \cup$ Label(N)
    $M$   $\leftarrow$   Train($L$)

Until:
    $U$   $=$   $\emptyset$ or *human stops*
```

Figure 1.19: Algorithm for sample selection [Hwa01]

This evaluation function f should denote the uncertainty of a parser for a particular sentence. [Hwa01, Hwa00] defines a particular evaluation function called f_{te}. If the tree entropy of a parser M for sentence u is defined as:

$$TE(u, M) = -\sum_{t \in \mathcal{T}} P(t \mid u, M) \cdot log_2 P(t \mid u, M)$$

Then the evaluation function that is used in Figure 1.2.4 is defined as:

$$f_{te}(u, M) = \frac{TE(u, M)}{log_2 |\mathcal{T}|}$$

In the experiment reported in [Hwa01], Michael Collins' parser was trained using this active learning technique by presenting the sentences from the Penn Treebank in the order selected by the sample selection technique as compared with simply using the entire labeled set without any selection. This experiment found that the parser could be trained a reduced number of annotated constituents (a reduction of 23%) for the same accuracy as using the entire labeled set.

1.2.5 Applications of statistical parsing

In this section we will give a survey of published results on application areas that improve on the state of the art performance by the use of statistical parsing. Parsing is a basic component of almost any natural language software application which is sometimes eliminated by the use of approximations typically using finite-state techniques. Advances in parsing should affect performance of a variety of natural language applications like named entity recognition, machine translation, information extraction and language modeling among others. In this section we take two examples: that of information extraction and language modeling and show how statistical parsing can have an impact on these applications.

Information extraction

Parsers trained on annotated parse trees that encode semantic values have been used for dialog projects like *ATIS*, *How May I Help You?* and *Communicator*. MUC-7 tasks like Template Element (TE) and Template Relations (TR) have been typically performed using shallow parsing methods using finite-state techniques due to a more expressive domain. Statistical Parsing has applied to this domain by BBN [MCF$^+$98] in the Template Element (TE) and the Template Relations (TR) task (see Figure 1.2.5).

The approach taken in [MCF$^+$98] is as follows

- Train a statistical parser on a general domain.
- Annotate a small set L of sentences with the expected output relations using the domain-specific MUC-7 template values.
- Parse L using the statistical parser to find parses consistent with the template annotation and combine the syntactic analysis and the semantic annotation based on a crossing bracket. measure. Figure 1.2.5 shows an example of such a combined parse tree.
- Retrain a new statistical parser that will produce the combined output.
- When the retrained parser is run on unseen text, it produces the desired domain-specific MUC-7 template values.

The performance of their system is shown in Figure 1.2.5.

```
<entity id="5" ent_type="person">
  <ent_name value="Nance"/>
  <ent_name value="Nancy Collins"/>
  <ent_descriptor value="paid consultant to ABC news"/>
</entity>

<entity id="6" ent_type="organization">
  <ent_name value="ABC"/>
  <ent_name value="ABC news"/>
</entity>

<relation type="employee_of">
  <entity arg="0" type="person" value="5"/>
  <entity arg="1" type="organization" value="6"/>
</relation>
```

Figure 1.20: Annotation corresponding to the Template Element (TE) and the Template Relations (TR) task.

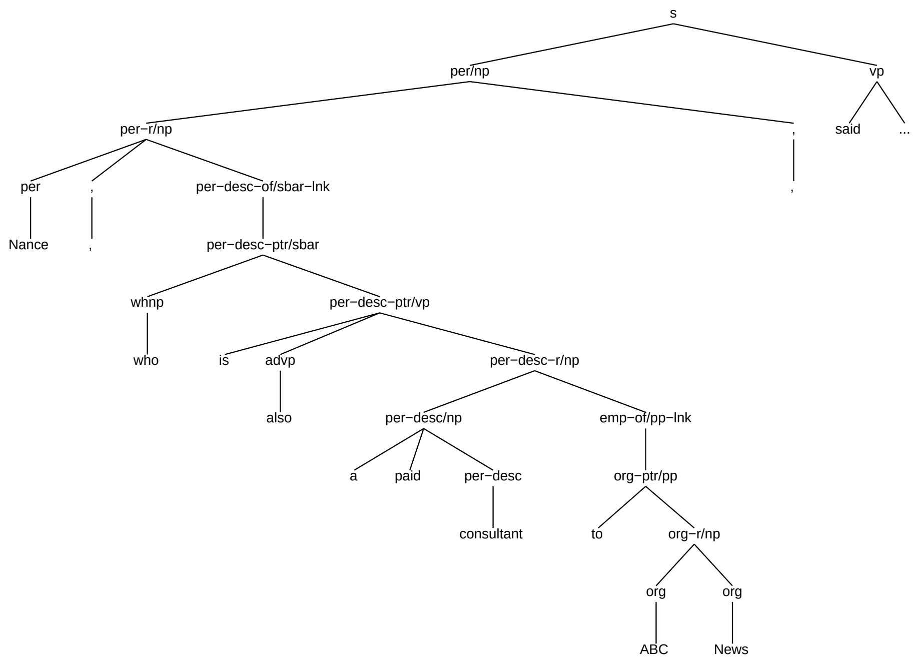


Figure 1.21: Combination of syntactic analysis and the MUC-7 template annotation

Task	Recall	Precision
Entities (TE)	83.0	84.0
Relations (TR)	64.0	81.0

Figure 1.22: Performance of the BBN MUC-7 system

Language modeling

Speech recognition requires a model for the most likely next token. This is done by building a language model: $P(t_k \mid t_1 \ldots t_{k-1})$. A statistical parser can be used as a language model simply by treating parsing as a task on a word lattice output of a speech recognizer. In this case spans in a string for a parser become states in a finite-state automaton.

Typically language models are built using a trigram context: $P(t_k \mid t_{k-3}, t_{k-2}, t_{k-1})$ which ignore any previous material that might be contingent on predicting the next token t_k. While there are other techniques such modeling the topic or modeling repetition of words, a useful technique might be to consider the syntactic context that has occurred in the history before t_k was produced. For example, Figure 1.2.5 shows anecdotal evidence to support the use of head-driven parsing to create a language model. For an input sentence *The contract ended with a loss of 7 cents after ...*, the bigram *ended, after* is a better predictor of the word *after* than the bigram *cents, after*.

When such a syntactic model was used as a language model using a head-driven parser in [CEJ+97, CJ98] the test-set perplexity was reduced as compared to a trigam model. Performance was even better when the syntactic model was interpolated with a trigram model. Figure 1.2.5 show the perplexity results.

In some cases reduction in perplexity does not always correspond to a reduction in the word-error rate which measures real improvement in transcription using a speech recognizer. [WK99] showed that the word-error rate can also be improved with the use of a syntactic language model (see Figure 1.2.5).

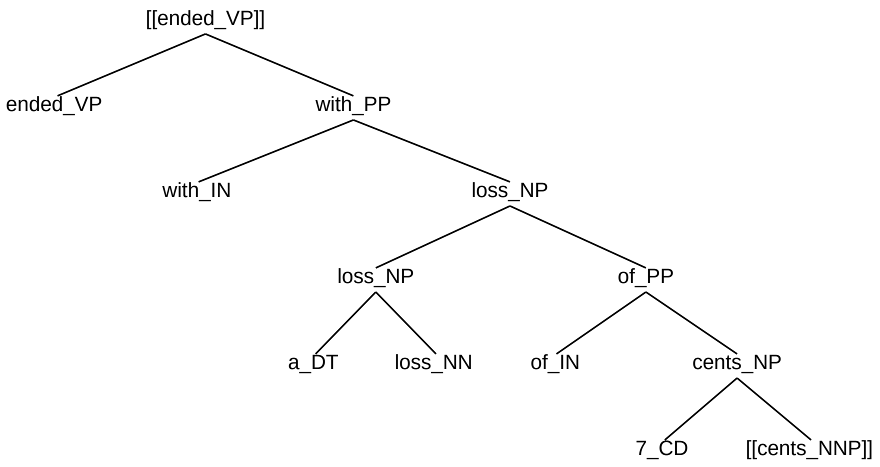


Figure 1.23: Utility of syntactic context for language modeling. The next word *after* is conditioned on the previous head word *ended* along with the previous word *cents*.

Iteration/Baseline	Test set Perplexity	Interpolation with 3-gram
Baseline Trigram	167.14	167.14
Iteration 0	167.47	152.25
Iteration 3	158.28	**148.90**

Figure 1.24: Perplexity results when using a syntactic language model

Model	Perplexity	WER
Baseline Trigram	79.0	38.5
N-gram + Syntactic	73.5	37.7

Figure 1.25: Word Error Rate results when using a syntactic language model

Chapter 2

Tree Adjoining Grammars and Statistical Parsing

2.1 Introduction

Tree-Adjoining Grammar (TAG) is a tree ‘manipulating’ system. Viewed generatively, it is is a tree rewriting system and viewed analytically, it can be viewed as a parser which parses a sentence into its component elementary trees. The elementary trees are lexically anchored in the case of a Lexicalized Tree-Adjoining Grammar (LTAG). The history of composition of these trees are a record of combining elementary trees with the operations of substitution and adjoining.

Large scale lexicalized TAG grammars have been constructed by hand for English at the University of Pennsylvania [XTA01] and French (at the TALANA group, University of Paris 7, France) and somewhat smaller ones for German (at DFKI, Saarbrücken, Germany), Korean (at UPenn), Chinese (at UPenn), and Hindi (at CDAC, Pune, India). The earliest stochastic variant of TAG was proposed by [Res92, Sch92]. LTAG grammars have been extracted from annotated corpora [XHPJ01, Xia01, Chi00, CVS00], which in turn have been used for statistical parsing [Chi00, Sar01]. The statistical parsing work done in TAGs emphasizes the use of lexicalized elementary trees and the recovery of the best derivation for a given sentence rather than the best parse tree.

The plan of this chapter is as follows. In Section 2.2 we will present a short introduction to

LTAG, pointing out specifically how LTAG arises in the natural process of lexicalization of context-free grammars (CFG). The resulting system is however, more powerful than CFGs, both in terms of weak generative capacity (string sets) and strong generative capacity (in terms of structural descriptions associated with the strings). In Sections 2.3 and 2.4 we will describe the stochastic models for TAGs and LTAGs and their application to statistical parsing. In Section 2.5 we show how a well-formed probabilistic generative process can be defined for stochastic TAGs. Finally, in Section 2.6 we will discuss some recent relevant work.

2.2 Tree-adjoining grammars

Tree-adjoining grammar (TAG) is a formal tree rewriting system. TAG and Lexicalized Tree-Adjoining Grammar (LTAG) have been extensively studied both with respect to their formal properties and to their linguistic relevance. TAG and LTAG are formally equivalent, however, from the linguistic perspective LTAG is the system we will be concerned with in this dissertation. We will often use these terms TAG and LTAG interchangeably.

The motivations for the study of LTAG are both linguistic and formal. The elementary objects manipulated by LTAG are structured objects (trees or directed acyclic graphs) and not strings. Using structured objects as the elementary objects of the formal system, it is possible to construct formalisms whose properties relate directly to the study of strong generative capacity (i.e., structural descriptions), which is more relevant to the linguistic descriptions than the weak generative capacity (sets of strings).

Each grammar formalism specifies a domain of locality, i.e., a domain over which various dependencies (syntactic and semantic) can be specified. It turns out that the various properties of a formalism (syntactic, semantic, computational, and even psycholinguistic) follow, to a large extent, from the initial specification of the domain of locality.

2.2.1 Domain of locality of CFGs

In a context-free grammar (CFG) the domain of locality is the one level tree corresponding to a rule in a CFG (Figure 2.1). It is easily seen that the arguments of a predicate (for example, the two arguments of *likes*) are not in the same local domain. The two arguments are distributed over

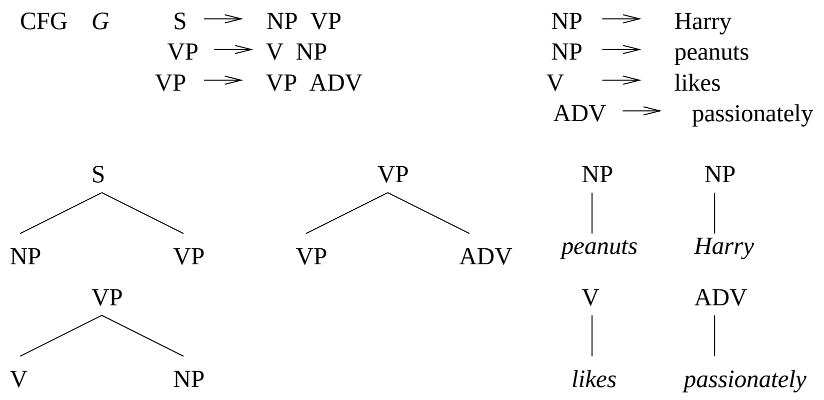


Figure 2.1: Domain of locality of a context-free grammar

the two rules (two domains of locality)– $S \rightarrow NP\ VP$ and $VP \rightarrow V\ NP$. They can be brought together by introducing a rule $S \rightarrow NP\ V\ VP$. However, then the structure provided by the VP node is lost[1]. We should also note here that not every rule (domain) in the CFG in (Figure 2.1) is lexicalized. The four rules on the right are lexicalized, i.e., they have a lexical anchor. The rules on the left are not lexicalized. The second and the third rules on the left are almost lexicalized, in the sense that they each have a preterminal category (V in the second rule and ADV in the third rule), i.e., by replacing V by *likes* and ADV by *passionately* these two rules will become lexicalized. However, the first rule on the left ($S \rightarrow NP\ VP$) cannot be lexicalized. Can a CFG be lexicalized, i.e., given a CFG, G, can we construct another CFG, G', such that every rule in G' is lexicalized and $T(G)$, the set of (sentential) trees (i.e., the tree language of G) is the same as the tree language $T(G')$ of G'? It can be shown that this is not the case (see [JS97]). Of course, if we require that only the string languages of G and G' be the same (i.e., they are weakly equivalent) then any CFG can be lexicalized. This follows from the fact that any CFG can be put in the Greibach normal form where each rule is of the form $A \rightarrow w\ B1\ B2\ ...\ Bn$ where w is a lexical item and the $B's$ are nonterminals. The lexicalization we are interested in requires the tree languages (i.e., the set of structural descriptions) be the same, i.e., we are interested in 'strong' lexicalization. To summarize, a CFG cannot be strongly lexicalized by a CFG. This follows from the fact that the domain of locality of CFG is a one level tree corresponding to a rule in the grammar. Note that

[1]Note that some approaches do not use the constituent domain of the VP. These approaches all have problems dealing with co-indexing and deletion of VPs as in cases of ellipsis and coordination.

there are two issues we are concerned with here – lexicalization of each elementary domain and the encapsulation of the arguments of the lexical anchor in the elementary domain of locality. The second issue is independent of the first issue. From the mathematical point of view the first issue, i.e., the lexicalization of the elementary domains of locality is the crucial one. We can obtain strong lexicalization without satisfying the requirement specified in the second issue (encapsulation of the arguments of the lexical anchor). Of course, from the linguistic point of view the second issue is very crucial. What this means is that among all possible strong lexicalizations we should choose only those that meet the requirements of the second issue. For our discussions in this dissertation we will assume that we always make such a choice.

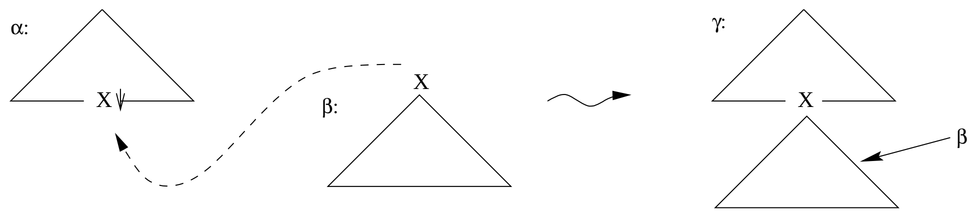


Figure 2.2: Substitution

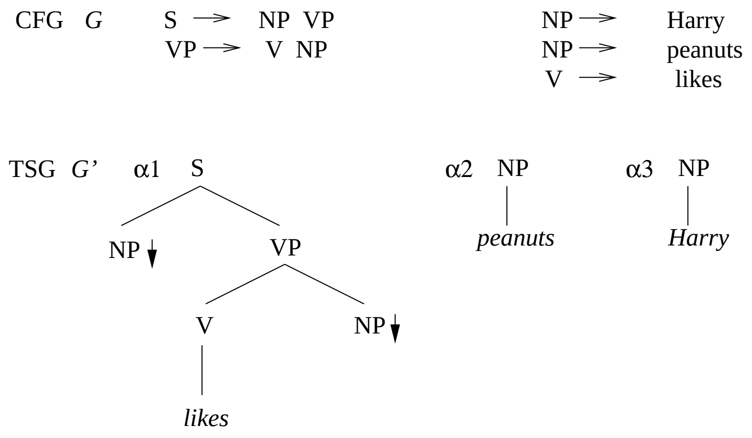


Figure 2.3: Tree substitution grammar

2.2.2 Lexicalization of CFGs

Now we can ask the following question. Can we strongly lexicalize a CFG by a grammar with a larger domain of locality? Figure 2.2 and Figure 2.3 show a tree substitution grammar where the elementary objects (building blocks) are the three trees in Figure 2.3 and the combining operation is the tree substitution operation shown in Figure 2.2. Note that each tree in the tree substitution grammar (TSG), G' is lexicalized, i.e., it has a lexical anchor. It is easily seen that G indeed strongly lexicalizes G. However, TSG's fail to strongly lexicalize CFG's in general. We show this by an example. Consider the CFG, G, in Figure 2.4 and a proposed TSG, G'. It is easily seen that although G and G' are weakly equivalent they are not strongly equivalent. In G, suppose we start with the tree α_1 then by repeated substitutions of trees in G' (a node marked with a vertical arrow denotes a substitution site) we can grow the right side of α_1 as much as we want but we cannot grow the left side. Similarly for α_2 we can grow the left side as much as we want but not the right side. However, trees in G can grow on both sides. Hence, the TSG, G', cannot strongly lexicalize the CFG, G [JS97].

- CFG G: $(r_1)\ S \rightarrow\ S\ S \quad (r_2)\ S \rightarrow\ a$
- Tree-substitution Grammar G':

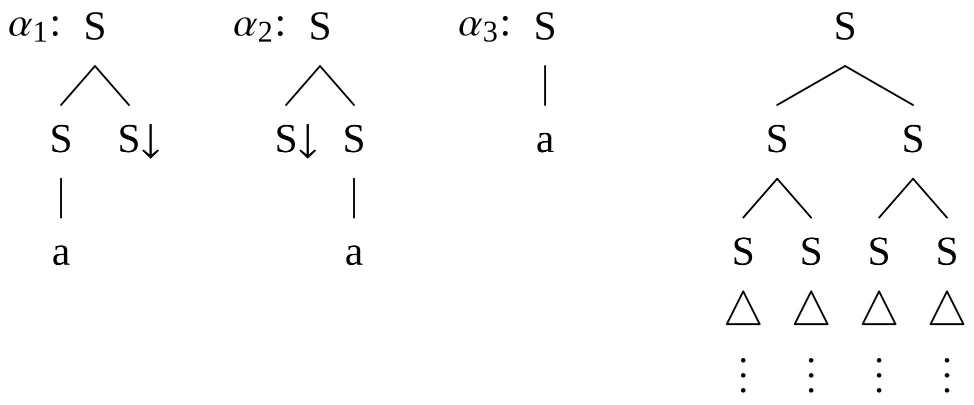


Figure 2.4: A tree substitution grammar for the given context-free grammar

We now introduce a new operation called 'adjoining' as shown in Figure 2.5. Adjoining involves splicing (inserting) one tree into another. More specifically, a tree β as shown in Figure 2.5 is inserted (adjoined) into the tree α at the node X resulting in the tree γ. The tree β, called an

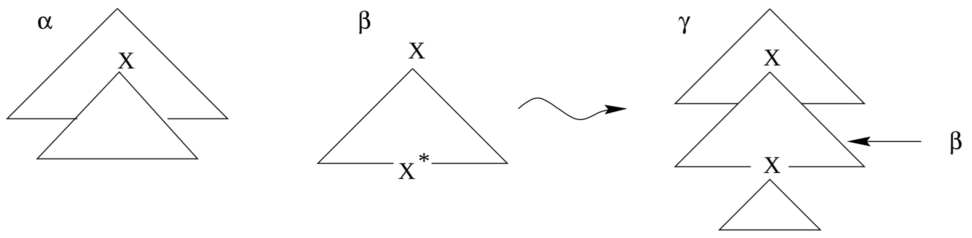


Figure 2.5: Adjoining

- CFG G: $(r_1)\ S \rightarrow S\ S \quad (r_2)\ S \rightarrow a$
- Tree-adjoining Grammar G'':

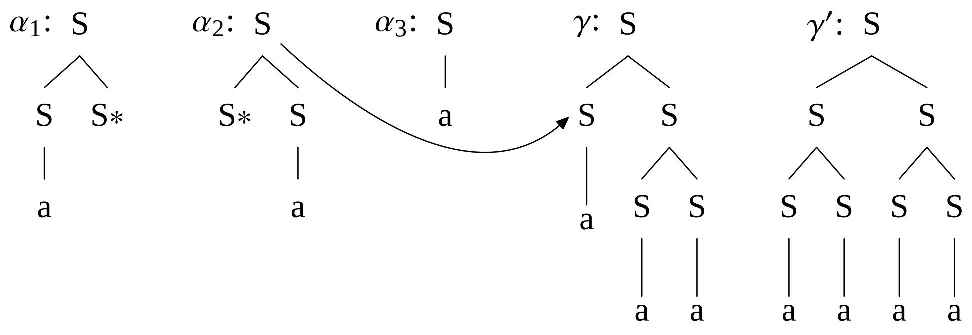


Figure 2.6: Adjoining arises out of lexicalization

auxiliary tree, has a special form. The root node is labeled with a nonterminal, say X and on the frontier there is also a node labeled X called the foot node (marked with *). There could be other nodes (terminal or nonterminal) nodes on the frontier of β, the nonterminal nodes will be marked as substitution sites (with a vertical arrow). Thus if there is another occurrence of X (other than the foot node marked with *) on the frontier of β it will be marked with the vertical arrow and that will be a substitution site. Given this specification, adjoining β to α at the node X in α is uniquely defined. Adjoining can also be seen as a pair of substitutions as follows: The subtree at X in α is detached, β is substituted at X and the detached subtree is then substituted at the foot node of β. A tree substitution grammar when augmented with the adjoining operation is called a tree-adjoining grammar (lexicalized tree-adjoining grammar because each elementary tree is lexically anchored). In short, LTAG consists of a finite set of elementary trees, each lexicalized with at least one lexical anchor. The elementary trees are either initial or auxiliary trees. Auxiliary trees have been defined already. Initial trees are those for which all nonterminal nodes on the frontier are substitution nodes. It can be shown that any CFG can be strongly lexicalized by an LTAG [JS97].

In Figure 2.6 we show a TSG, G', augmented by the operation of adjoining, which strongly lexicalizes the CFG, G. Note that the LTAG looks the same as the TSG considered in Figure 2.4. However, now trees α_1 and α_2 are auxiliary trees (containing a foot node marked with *) that can participate in adjoining. Since adjoining can insert a tree in the interior of another tree it is possible to grow both sides of the tree α_1 and tree α_2, which was not possible earlier with substitution alone. In summary, we have shown that by increasing the domain of locality we have achieved the following: (1) lexicalized each elementary domain, (2) introduced an operation of adjoining, which would not be possible without the increased domain of locality (note that with one level trees as elementary domains adjoining becomes the same as substitution since there are no interior nodes to be operated upon), and (3) achieved strong lexicalization of CFGs.

2.2.3 Lexicalized tree-adjoining grammars

Rather than giving formal definitions for LTAG and derivations in LTAG (see [VS87] for a formal definition of a TAG derivation) we will give a simple example to illustrate some key aspects of LTAG. We show some elementary trees of a toy LTAG grammar of English. Figure 2.7 shows two elementary trees for a verb such as *likes*. The tree α_1 is anchored on *likes* and encapsulates the

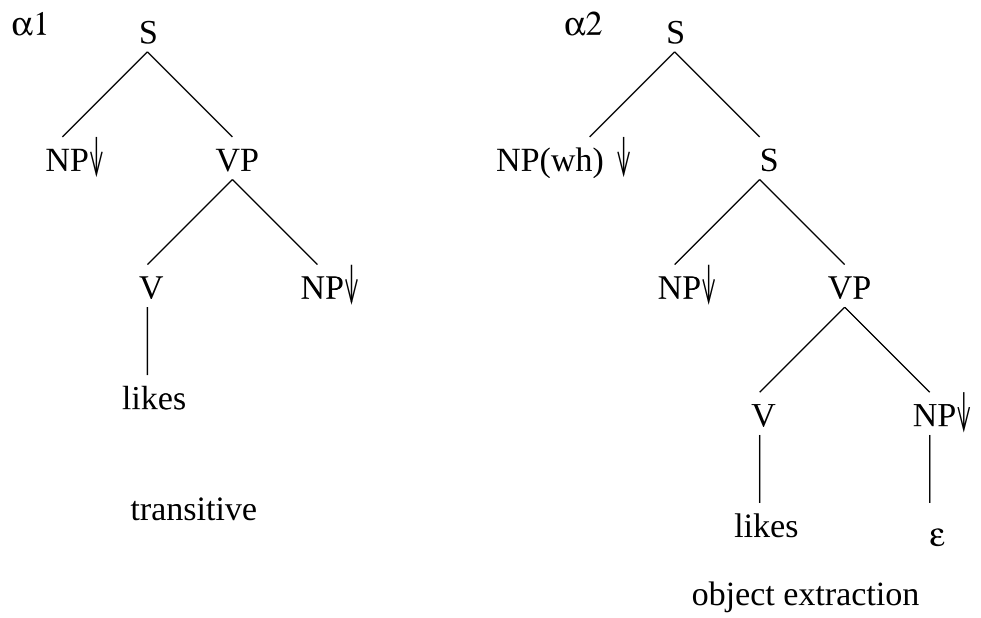


Figure 2.7: LTAG: Elementary trees for *likes*

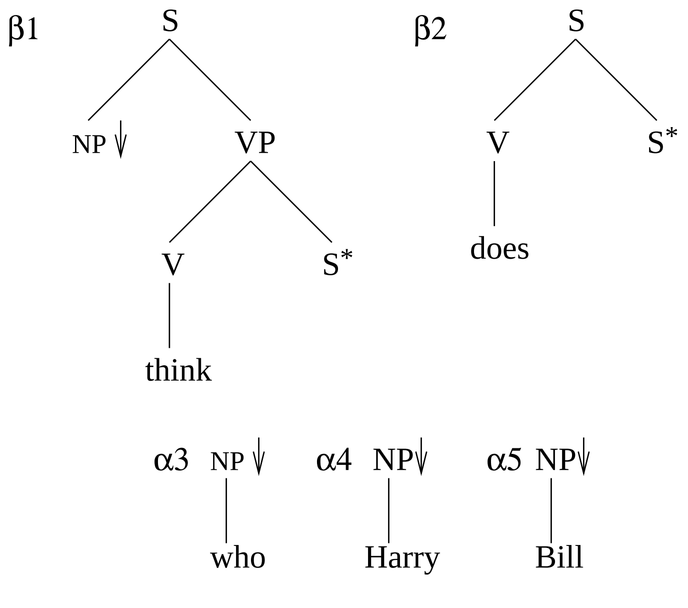


Figure 2.8: LTAG: Sample elementary trees

two arguments of the verb. The tree α_2 corresponds to the object extraction construction. Since we need to encapsulate all the arguments of the verb in each elementary tree for *likes*, for the object extraction construction, for example, we need to make the elementary tree associated with *likes* large enough so that the extracted argument is in the same elementary domain. Thus, in principle, for each 'minimal' construction in which *likes* can appear (for example, subject extraction, topicalization, subject relative, object relative, passive, etc.) there will be an elementary tree associated with that construction. By 'minimal' we mean when all recursion has been factored away. This factoring of recursion away from the domain over which the dependencies have to be specified is a crucial aspect of LTAGs as they are used in linguistic descriptions. This factoring allows all dependencies to be localized in the elementary domains. In this sense, there will, therefore, be no long distance dependencies as such. They will all be local and will become long distance on account of the composition operations, especially adjoining.

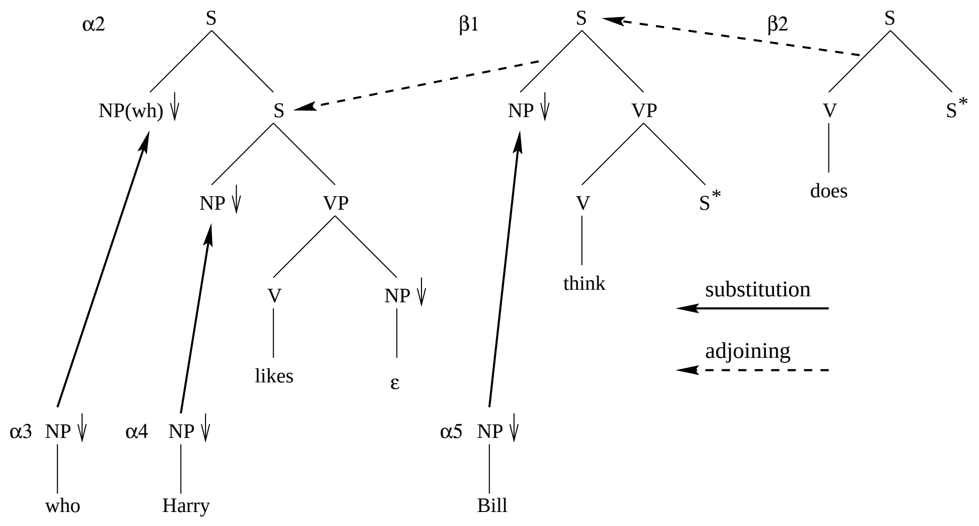


Figure 2.9: LTAG derivation for *who does Bill think Harry likes*

Figure 2.8 shows some additional trees. Trees α_3, α_4, and α_5 are initial trees and trees β_1 and β_2 are auxiliary trees with foot nodes marked with *. A derivation using the trees in Figure 2.7 and Figure 2.8 is shown in Figure 2.9. The trees for *who* and *Harry* are substituted in the tree for *likes* at the respective *NP* nodes, the tree for *Bill* is substituted in the tree for *think* at the *NP* node, the tree for *does* is adjoined to the root node of the tree for *think* tree (adjoining at the root node is a special case of adjoining), and finally the derived auxiliary tree (after adjoining β_2 to β_1) is adjoined to the indicated interior S node of the tree α_2. This derivation results in the **derived tree** for *who does Bill think Harry likes* as shown in Figure 2.10. Note that the dependency between *who* and the complement *NP* in α_2 (local to that tree) has been stretched in the derived tree in Figure 2.10. This tree is the conventional tree associated with the sentence.

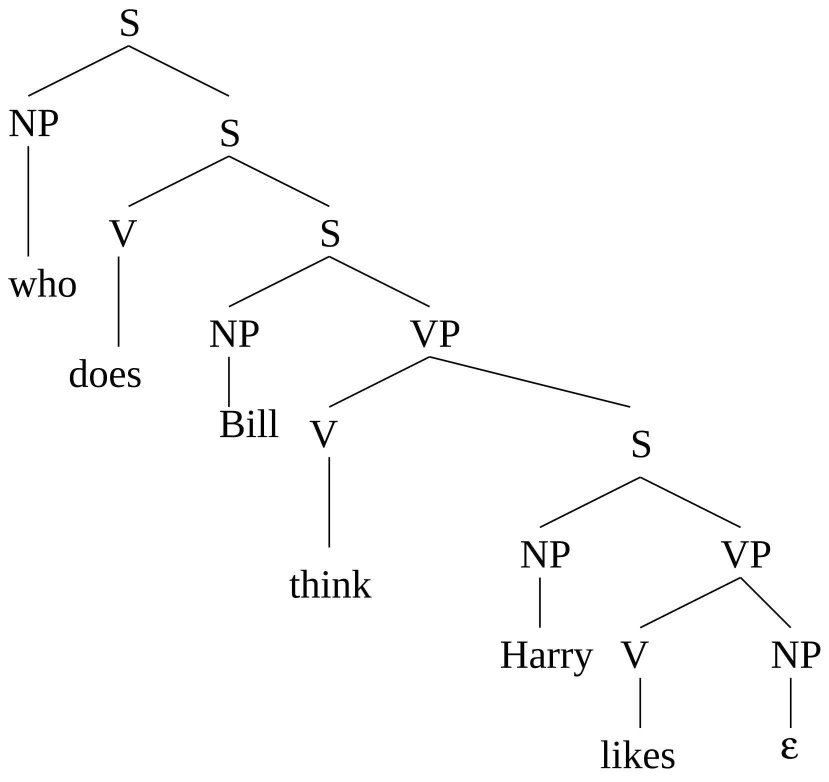


Figure 2.10: LTAG derived tree for *who does Bill think Harry likes*

However, in LTAG there is also a derivation tree, the tree that records the history of composition of the elementary trees associated with the lexical items in the sentence. This derivation tree is shown in Figure 2.11. The nodes of the tree are labeled by the tree labels such as α_2 together with the lexical anchor.[2] The derivation tree is the crucial derivation structure for LTAG. We can obviously build the derived tree from the derivation tree. For semantic computation the derivation tree (and not the derived tree) is the crucial object. Compositional semantics is defined

[2]The derivation trees of LTAG have a close relationship to the dependency trees, although there are some crucial differences; however, the semantic dependencies are the same. See [RJ95] for more details.

on the derivation tree. The idea is that for each elementary tree there is a semantic representation associated with it and these representations are composed using the derivation tree.

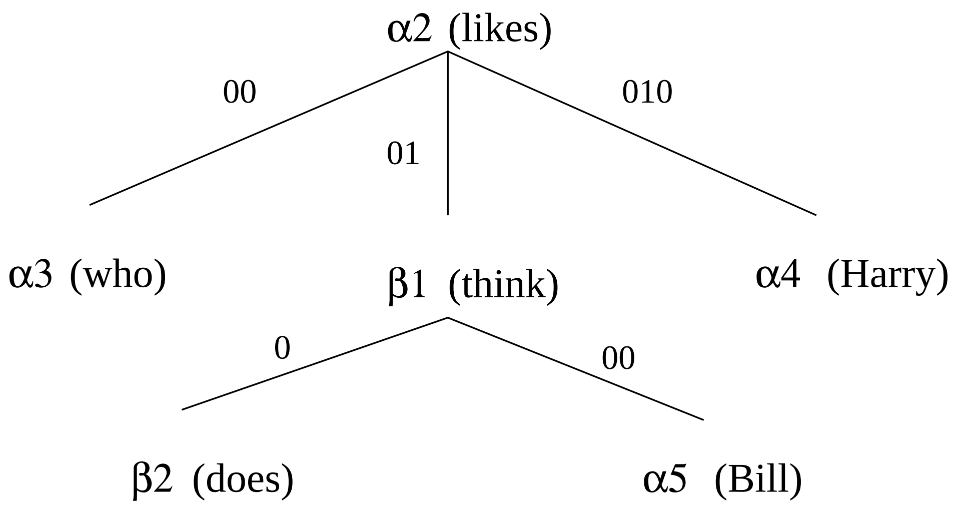


Figure 2.11: LTAG derivation tree

It is important to note that the node addresses which are part of the definition of a TAG derivation tree represents important information. In many introductions to TAG, the derivation tree is equated with a dependency tree implying that a compositional semantics for a TAG grammar is same as what one would define on a dependency tree. This is not the case. The node addresses in a derivation tree add crucial information unavailable to dependency trees. Let us take strong lexicalization where the arguments of the lexical anchor (the predicate) are localized within an elementary tree. This captures the fact that the predicate-argument structure for the verb *likes* in the sentence: *John likes soccer* is represented in the elementary tree for the verb *likes* (as in Figure 2.7). However, the elementary tree for *likes* does more than that. It also sets up certain structural relationships, as for example, the relationship of agreement between the subject NP substitution node and the VP node in the tree it anchors. This agreement relationship is local to the elementary tree and is used to capture some apparently non-local relationships in the TAG view of elementary trees. For example, the relationship between the subject NP and the VP node in the *like* tree captures the agreement facts between sentences such as *He seems to like soccer* and *They seem to like soccer*. This contrast can be explained only when you consider the contribution of the node address in the derivation tree.

Another contribution of the node addresses in the TAG derivation tree is that of scope. In a sentence like *John thinks Mary seems to like soccer*, the different location of adjuction of the *thinks* tree (at the S node) and the *seems* tree (at the VP node) provides the semantics as being the correct one *think(john, seem(like(mary, soccer)))* rather than the incorrect *seem(think(john, like(mary,soccer)))*.

2.2.4 Some important properties of LTAG

The two key properties of LTAG are (1) extended domain of locality (EDL) (for example, as compared to CFG), which allows (2) factoring recursion from the domain of dependencies (FRD), thus making all dependencies local. All other properties of LTAG (mathematical, linguistic, and even psycholinguistic) follow from EDL and FRD. TAGs (LTAGs) belong to the so-called class of mildly context-sensitive grammars [Jos85]. Context-free languages (CFL) are properly contained in the class of languages of LTAG, which in turn are properly contained in the class of context-sensitive languages. There is a machine characterization of TAG (LTAG), called embedded pushdown automaton (EPDA) [VS87], i.e., for every TAG language there is an EPDA which corresponds to this (and only this) language and the language accepted by any EPDA is a TAG language. EPDAs have been used to model some psycholinguistic phenomena, for example, processing crossed dependencies and nested dependencies have been discussed in [Jos90]. With respect to formal properties, the class of TAG languages enjoys all the important properties of CFLs, including polynomial parsing (with complexity $O(n^6)$). Under certain restrictions on adjunction which disallow the creation of wrapping auxiliary trees (called Tree Insertion Grammars), we can reduce the worst-case parsing complexity to that of CFGs: $O(n^3)$.

Large scale wide coverage grammars have been built using LTAG, the XTAG system (LTAG grammar and lexicon for English and a parser) being the largest so far (for further details see [XTA01]. In the XTAG system, each node in each LTAG tree is decorated with two feature structures (top and bottom feature structures), in contrast to the CFG based feature structure grammars. This is necessary because adjoining can augment a tree internally, while in a CFG based grammar or even in a tree substitution grammar a tree can be augmented only at the frontier. It is possible to define adjoining and substitution (as it is done in the XTAG system) in terms of appropriate unifications of the top and bottom feature structures. Because of FRD (factoring recursion from the

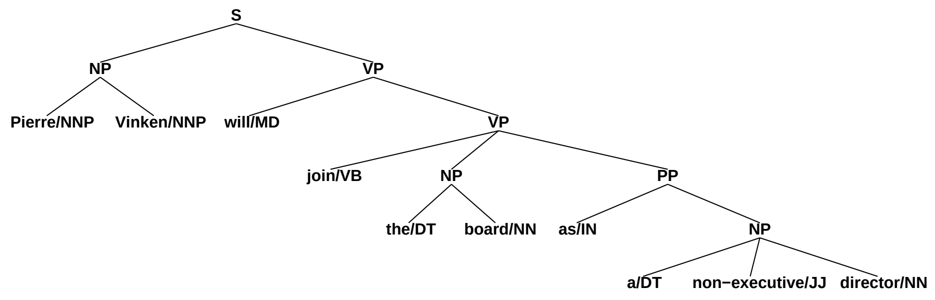


Figure 2.12: Parse tree for an example sentence taken from the Penn Treebank

domain of dependencies), there is no recursion in the feature structures. Therefore, in principle, feature structures can be eliminated. However, they are the conventional vehicle for specifying linguistic constraints. Such constraints on substitution and adjoining are modeled via these feature structures [VS87]. This method of manipulating feature structures is a direct consequence of the extended domain of locality of LTAG.

2.3 Statistical Parsing with Tree Adjoining Grammars

Before we provide a formal definition for stochastic Tree Adjoining Grammars, we provide some motivation for its use in statistical parsing. In building a statistical parser our task is to find the sub-parts or constituents for naturally occuring sentences (e.g. see Figure 2.12).

These constituents are recursively embedded within one another and hence we decompose the entire structure assigned to a sentence into smaller parts. A common theory underlying this decomposition is that of context-free grammars (CFGs). For example, for the constituency structure where the non-terminals have been updated with lexicalized information as shown in Figure 2.14, we extract context-free rules of the kind shown in Figure 2.15. In order for the phrase structure to be sensitive to the lexical information each constituent has a *head* word that represents it in the phrase structure.

However, the right hand side of each context-free rule has multiple constituent elements each with its own head word. Therefore, rules can be lexicalized with several words at once. Since

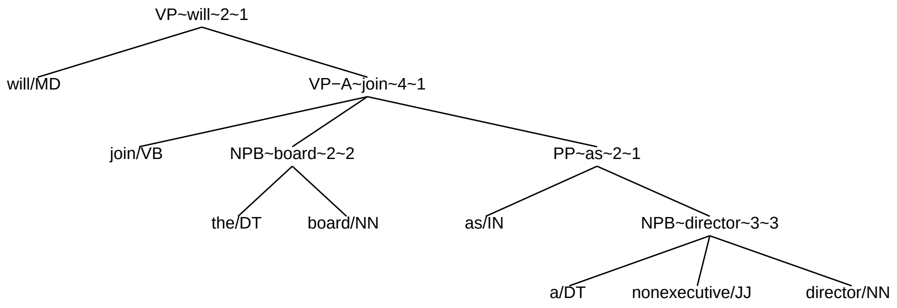


Figure 2.13: Typical head-lexicalization by heuristic rules for an example sentence from the Penn Treebank

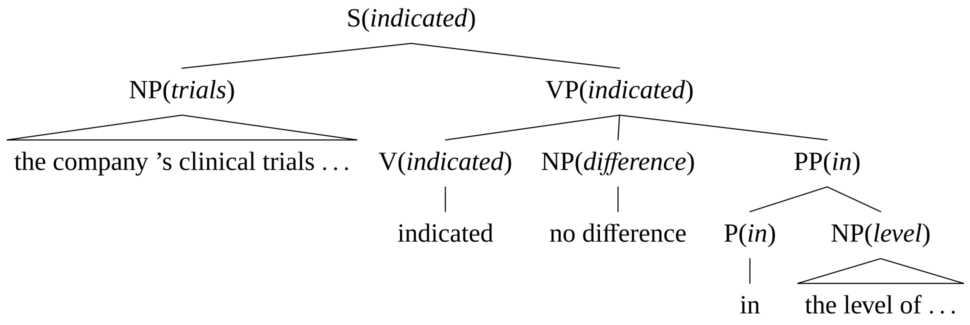


Figure 2.14: A parse tree from the Treebank for the sentence: *the company 's clinical trials of both its animal and human-based insulins indicated no difference in the level of hypoglycemia between users of either product*

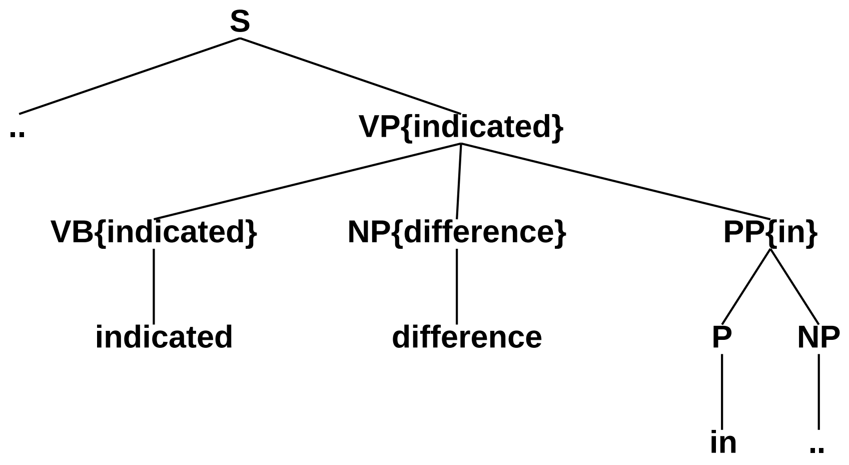


Figure 2.15: Context-free rule or a tree of depth= 1.

words occur sparsely in any corpus, we are quite unlikely to see three or more words that have occurred together in the training data again in the test data. For this reason, standard CFG-based statistical parser impose a further independence assumption that the right hand side of each rule is produced by a Markov process. The generation of each CFG rule is shown in Figure 2.16. Each dependent of the head of the VP phrase which in this case is the verb *indicated* is generated independently of the other dependents on the head. For example, the NP headed by *difference* and the PP headed by *in* are attached to the verb independently of each other. There is an added symbol *STOP* to make well-defined 0-th order Markov process.

However, the independence assumptions underlying the decomposition shown in Figure 2.16 are violated in the corpus. Let us consider subtrees of depth greater than 1 and their empirical distribution in the Penn Treebank WSJ corpus. In subtrees where a VP node dominates another VP node as shown in Figure 2.17 we see a marked difference in the expected attachment of a PP to the various VP positions. The PP is far more likely to attach the the lower VP rather than to the higher VP. This additional fact about the distribution of subtrees of phrase structure has to be added into a model which makes the kind of independence assumptions shown in Figure 2.16. Figure 2.17

- Full context-free rule:
  VP(*indicated*) → V-hd(*indicated*) NP(*difference*) PP(*in*)
- Each rule is generated in three steps (Collins 1999):
  1. Generate head daughter of LHS: VP(*indicated*) → V-hd(*indicated*)
  2. Generate non-terminals to *left* of head daughter: STOP . . . V-hd(*indicated*)
  3. Generate non-terminals to *right* of head daughter:
     - V-hd(*indicated*) . . . NP(*difference*)
     - V-hd(*indicated*) . . . PP(*in*)
     - V-hd(*indicated*) . . . STOP

Figure 2.16: Independence assumptions that decompose the right hand side of the CFG rule.

also shows that the independence assumptions discussed earlier can be very beneficial. Ignoring an optional PP can help us generalize to new cases in the test data even if we have not seen a particular right hand side of a CFG rule used in the test data.

Other independence assumptions are noted in (Collins 1999) that are violated in the Treebank are listed below. These were folded into the models given in (Collins 1999) as subsequent modifications of the original model, each change carefully removing the independence assumptions in particular syntactic contexts.

- Markov grammar assumption violated in cases of coordination.
  e.g. `NP and NP`; `VP and VP`
- Processing facts like attach low in general.
- Also, English parse trees are generally right branching due to SVO structure.
- Language specific features are used heavily in the statistical model for parsing: cf. (Haruno et al. 1999)

The question is whether we can retain the advantages of the independence assumptions shown in Figure 2.16 while at the same time being sensitive to the empirical facts about subtrees shown in Figure 2.17.

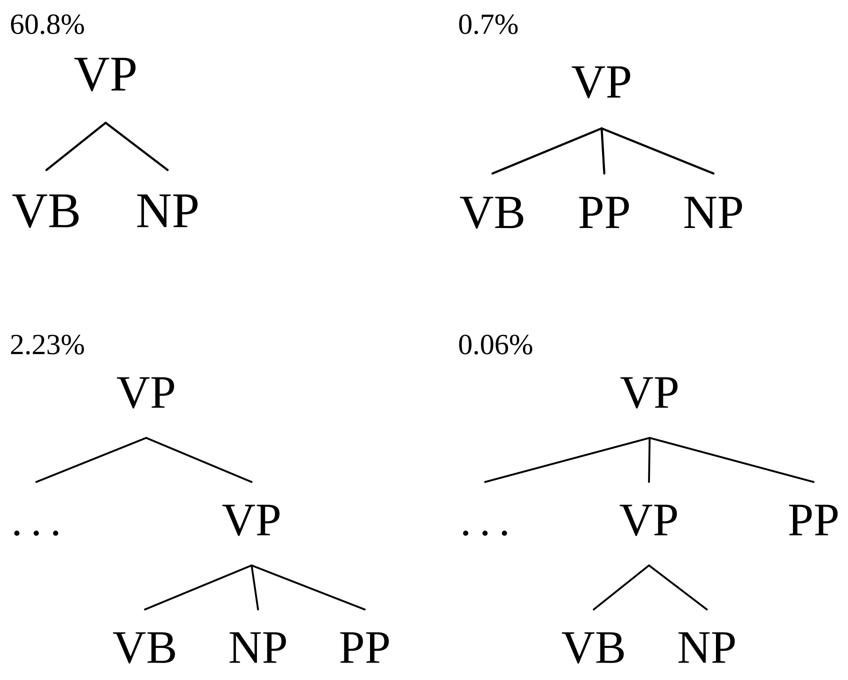


Figure 2.17: Independence assumptions are violated by only considering trees of depth= 1.

CFG rules can be viewed as trees of depth 1. A formalism that uses trees of depth greater than 1 would be able to capture the facts since it will be more sensitive to various possible attachment points along the spine of the tree. TAG is one such formalism.

Constructing a derivation in LTAG proceeds as follows: each word in a sentence is assigned a set of trees. Each of these trees assigned to the words in the sentence can combine with each other to produce a derivation from which an ordinary constituency phrase structure tree is produced. The yield of this tree is the input sentence.

Hence, TAG is an alternative method to the modeling of bilexical dependencies. For example, see Figure 2.18.

Each combination of a pair of trees is given a probability. For example, Figure 2.19 shows the probability model for substituting a tree for a non-terminal at the frontier of another tree.

Non-terminals that occur internal to each tree can also be rewritten by the operation of adjunction which replaces the non-terminal with a tree. For example, the adjunction of a tree into a node η is shown in Figure 2.20. The additional term that is missing from the substitution model is the

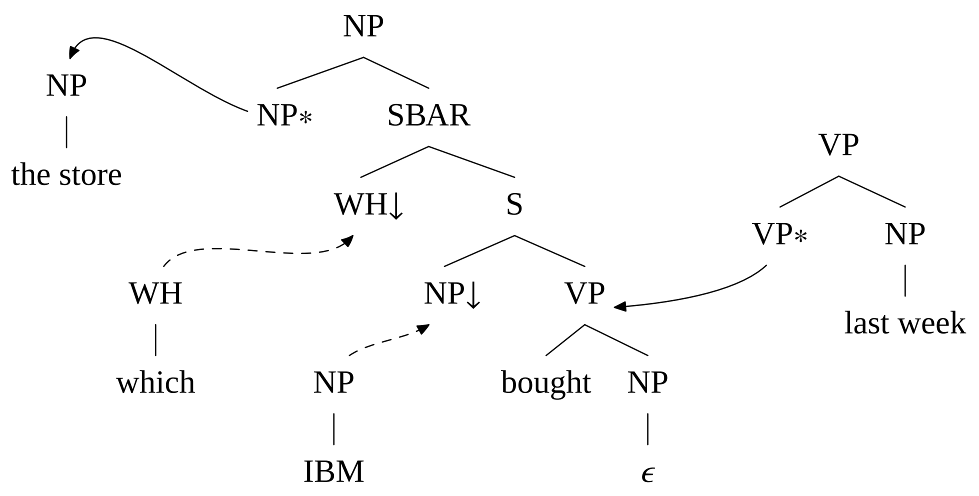
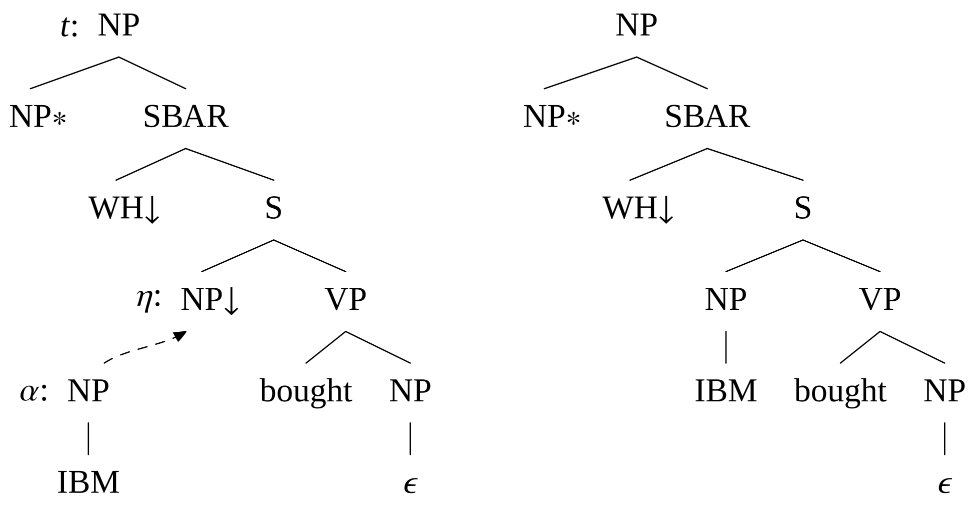
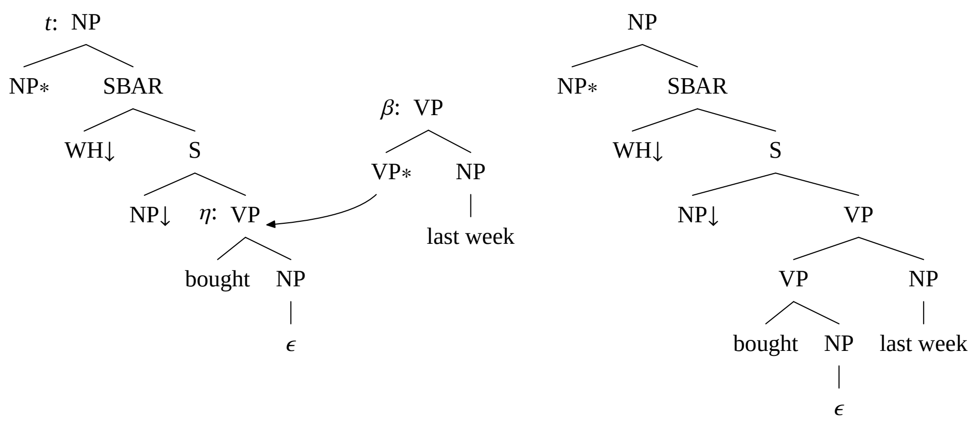


Figure 2.18: Alternative modeling of bilexical dependencies using a stochastic TAG.



$$\sum_{t'} \mathcal{P}(t, \eta \rightarrow t') = 1$$

Figure 2.19: TAG: Substitution



$$\mathcal{P}(t, \eta \rightarrow NA) + \sum_{t'} \mathcal{P}(t, \eta \rightarrow t') = 1$$

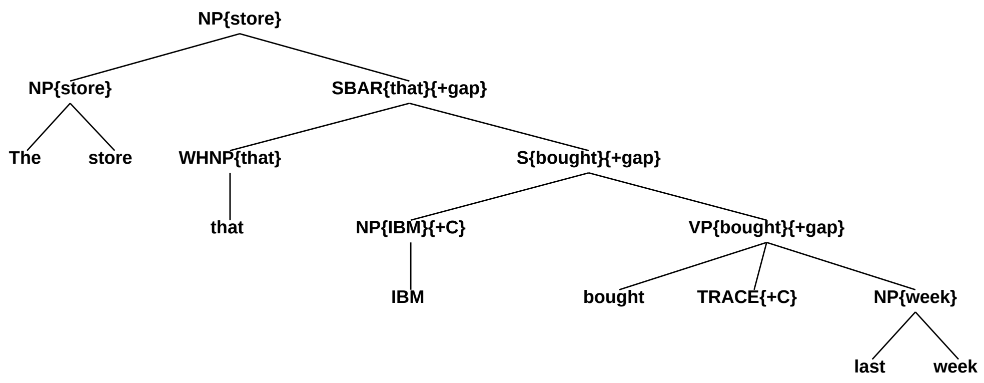
Figure 2.20: TAG: Adjunction

probability that the non-terminal is not re-written during the derivation (called the null-adjunction or *NA* probability).

TAGs have the following attractive properties as a framework for statistical parsing:

- Locality and independence assumptions are captured elegantly.
- Simple and well-defined probability model.
- Parsing can be treated in two steps:
  1. Classification: structured labels (elementary trees) are assigned to each word in the sentence.
  2. Attachment: the elementary trees are connected to each other to form the parse.

In addition, statistical parsers that are serious about interacting with a component that links the sentence to a meaning have to produce more than the phrase structure of each sentence. A more embellished parse in which phenomena such as predicate-argument structure, subcategorization and movement are given a probabilistic treatment is often expected from a parser. A CFG parser has to deal with such extensions by appealing to some kind of ‘feature’ based account as shown in Figure 2.21. Note that such additional information already forms part of every lexicalized tree in the TAG framework as shown in Figure 2.18.



NP	$\rightarrow$	Δ_{stop}	NP{+H}	SBAR{+gap}		Δ_{stop}
SBAR{+gap}	$\rightarrow$	Δ_{stop}	WHNP	S{+H}{+C}{+gap}		Δ_{stop}
S{+gap}	$\rightarrow$	Δ_{stop}	NP{+C}	SBAR{+H}{+gap}		Δ_{stop}
VP{+gap}	$\rightarrow$	Δ_{stop}	VB{+H}	TRACE{+C}	NP	Δ_{stop}

Figure 2.21: Features used in Collins' bi-lexical CFG parser ([Col97])

2.4 Stochastic Tree Adjoining Grammars

[Res92] provided some early motivation for a stochastic version of Tree Adjoining Grammars and gave a formal definition of stochastic TAG. Simultaneously, [Sch92] also provided an identical stochastic version of TAG and also extended the Inside-Outside algorithm for CFGs [LY90] to stochastic TAGs. [Sch92] also performed experiments to show that a stochastic TAG can be learnt from the ATIS corpus.

A stochastic LTAG derivation proceeds as follows [Sch92, Res92]. An initial tree is selected with probability P_i and subsequent substitutions are performed with probability P_s and adjunctions are performed with probability P_a.

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

$$\sum_{\tau} P_i(\tau) = 1$$

Each subsequent substitution or adjunction occurs independently. For possible substitutions defined by the grammar:

$$\sum_{\tau'} P_s(\tau, \eta \rightarrow \tau') = 1$$

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

$$P_a(\tau, \eta \rightarrow \textsc{na}) + \sum_{\tau'} P_a(\tau, \eta \rightarrow \tau') = 1$$

$P_a(\tau, \eta \rightarrow \textsc{na})$ is the probability that there is no adjunction (NA) at node η in τ.

Each LTAG derivation $\mathcal{D}$ is built starting from some initial tree α. Let us consider the probability of a derivation $\mathcal{D}$ which was constructed using p substitutions and q adjunctions and r internal nodes which had no adjunction. If we assume, for simplicity that each elementary tree is lexicalized by exactly one word, then the length of the sentence $n = p + q + 1$.

$$\begin{aligned} Pr(\mathcal{D}, w_0 \ldots w_n) = & \\ & P_i(\alpha, w_i) \times \prod_p P_s(\tau, \eta, w \rightarrow \tau', w') \times \\ & \prod_q P_a(\tau, \eta, w \rightarrow \tau', w') \times \\ & \prod_r P_a(\tau, \eta, w \rightarrow \textsc{na}) \end{aligned} \tag{2.1}$$

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.

P_i and P_s can be written as the following conditional probabilities which can be estimated from the training data. For further details about decomposing these probabilities further to make parsing and decoding easier and details about prior probabilities see [Chi01].

$$P_i(\alpha, w \mid TOP)$$
$$P_s(\alpha, w' \mid \tau, \eta, w)$$
$$P_a(\beta, w' \mid \tau, \eta, w)$$

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 $\mathcal{D}_{\text{BEST}}$ given a sentence S. The value for (2.2) is computed using the Equation 2.1.

$$\begin{aligned}\mathcal{D}_{\text{BEST}} &= \arg\max_{\mathcal{D}} \Pr(\mathcal{D} \mid S) \\ &= \arg\max_{\mathcal{D}} \frac{\Pr(\mathcal{D}, S)}{\Pr(S)} \\ &= \arg\max_{\mathcal{D}} \Pr(\mathcal{D}, S) \qquad (2.2)\end{aligned}$$

The particular definition of a stochastic TAG is by no means the only way of defining a probabilistic grammar formalism with TAG. There have been some variants from the standard model that have been published since the original stochastic TAG papers.

For example, the restriction of one adjunction per node could be dropped and a new variant of standard TAG can be defined which permits arbitrary number of modifications per node. This variant was first introduced by [SS92, SS94]. Tree Insertion Grammar [SW95] is a variant of TAG where the adjoining operation is restricted in a certain way and this restricted operation is named insertion. TIGs are weakly equivalent to CFGs but they can produce structural descriptions that are not obtainable by any CFG.

A stochastic version of insertion [SW96] was defined in the context of Tree Insertion Grammar. In this model, multiple trees can be adjoined to the left and to the right of each node with the following probabilities:

$$P_{la}(\tau, \eta \rightarrow \text{NA}_l) + \sum_{\tau'} P_{la}(\tau, \eta \rightarrow \tau') = 1$$

$$P_{ra}(\tau, \eta \rightarrow \text{NA}_r) + \sum_{\tau'} P_{ra}(\tau, \eta \rightarrow \tau') = 1$$

There are many other probability measures that can be used with TAG and its variants. One can easily go beyond the bi-lexical probabilities that have been the main focus in this chapter to probabilities that invoke greater amounts of structural or lexical context. [CW97], for example, gives some additional probability models one might consider useful when using TAGs.

2.5 Applying probability measures to Tree Adjoining Languages

In this section we look at some formal properties of stochastic TAGs. To gain some intuition about probability assignments to tree adjoining 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 [VS87], p. 104). An example of such a function ψ is a simple Poisson distribution (2.3), which in fact was also used as the counterexample in [BT73] for CFGs, since CFGs also have the constant growth property.

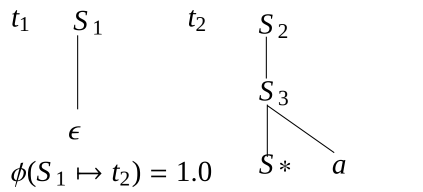
$$\psi(a^n b^n c^n d^n) = \frac{1}{e \cdot n!} \tag{2.3}$$

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. Consider the probabilistic TAG shown in (2.4)[3].



$$\phi(S_1 \mapsto t_2) = 1.0$$
$$\phi(S_2 \mapsto t_2) = 0.99$$
$$\phi(S_2 \mapsto nil) = 0.01$$
$$\phi(S_3 \mapsto t_2) = 0.98$$
$$\phi(S_3 \mapsto nil) = 0.02 \tag{2.4}$$

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 be blamed. 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.

2.5.1 Conditions for Consistency

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

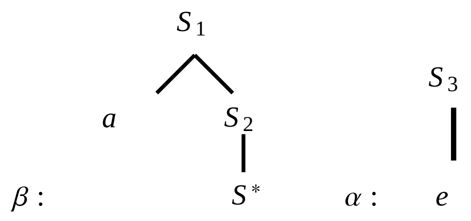
$$\sum_{v \in L(G)} \Pr(v) = 1 \tag{2.5}$$

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. Note that this is a very general definition and consistency can be defined for all well-defined generative processes [Har63].

[3]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.

To show that a given probabilistic TAG is consistent we can exploit context-free nature of TAG derivations and exploit the existing result by [BT73, Wet80] which provides conditions under which stochastic CFGs can be shown to be consistent[4].

We explain the alternative method using an example. Consider a PTAG $G = \langle\{\beta, \alpha\}, \phi\rangle$ with trees defined in (2.6) and parameter values given in (2.7).



(2.6)

$$
\begin{aligned}
\phi(S_1 \rightarrow \beta) &= 0.99 \\
\phi(S_1 \rightarrow \epsilon) &= 0.01 \\
\phi(S_2 \rightarrow \beta) &= 0.98 \\
\phi(S_2 \rightarrow \epsilon) &= 0.02 \\
\phi(S_3 \rightarrow \beta) &= 1.0 \\
\phi(S_3 \rightarrow \epsilon) &= 0.0
\end{aligned}
\tag{2.7}
$$

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 ?*

The Booth and Thompson result can be stated as follows:

Theorem 2.5.1 *A probabilistic grammar is consistent if the* spectral radius $\rho(\mathcal{M}) < 1$, *where $\mathcal{M}$ is the stochastic expectation matrix computed from the context-free grammar. [BT73, Sou74]*

This theorem provides a way to determine whether a grammar is consistent. All we need to do is compute the spectral radius of the expectation matrix $\mathcal{M}$, which defines the expected number of times each CFG rule will be seen in a corpus of parse trees generated by the stochastic CFG. The spectral radius 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]Thanks to Steve Abney for discussions on this topic.

In (2.8) we show a set of productions (rules) P constructed from the PTAG G designed to show the derivations possible in G.

$$\begin{array}{rcl} \alpha & \rightarrow & S_3 \\ S_3 & \rightarrow & \beta \mid \epsilon \\ \beta & \rightarrow & S_1 S_2 \\ S_1 & \rightarrow & \beta \mid \epsilon \\ S_2 & \rightarrow & \beta \mid \epsilon \end{array} \tag{2.8}$$

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. 2.9.

$$\sum_{(X \rightarrow Y) \in P} \mathcal{P}(X \rightarrow Y \mid X) = 1 \tag{2.9}$$

The rule probabilities for the CFG in (2.8) derived from the PTAG parameters in (2.7) are shown in (2.10).

$$\begin{array}{rcl} \mathcal{P}(\alpha \rightarrow S_3 \mid \alpha) & = & \sum_X \mathcal{P}(S_3 \rightarrow X \mid S_3) = 1.0 \\ \mathcal{P}(S_3 \rightarrow \beta \mid S_3) & = & 1.0 \\ \mathcal{P}(S_3 \rightarrow \epsilon \mid S_3) & = & 0.0 \\ \mathcal{P}(\beta \rightarrow S_1 S_2 \mid \beta) & = & \sum_X \mathcal{P}(S_1 \rightarrow X \mid S_1) \times \sum_Y \mathcal{P}(S_2 \rightarrow Y \mid S_2) = 1.0 \\ \mathcal{P}(S_1 \rightarrow \beta \mid S_1) & = & 0.99 \\ \mathcal{P}(S_1 \rightarrow \epsilon \mid S_1) & = & 0.01 \\ \mathcal{P}(S_2 \rightarrow \beta \mid S_2) & = & 0.98 \\ \mathcal{P}(S_2 \rightarrow \epsilon \mid S_2) & = & 0.02 \end{array} \tag{2.10}$$

Based on these probabilities we can compute the expectation matrix $\mathcal{M}$ using the method defined in [BT73].

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

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 inconsistent.

A formal justification for this method of showing whether a given stochastic TAG is consistent is given in Appendix A (also see [Sar98]) by showing a reduction of the TAG derivation process to a multitype Galton-Watson branching process [Har63].

2.5.2 Inside-Outside Probabilities and Prefi x Probabilities

Embedding Stochastic 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:[5]

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

Let input string $w = a_1 \ldots a_n$. 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.[6]

Init cases:

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

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

[5]Or perhaps

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

[6]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 $0 \leq i, j \leq n$, [7]

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

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.

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

In a similar fashion, one can derive the equations for the *outside* probability computation. Since the outside probabilities are considerably more complex than the inside probabilities, the reader is referred to the paper [Sch92] where the full set of equations is presented in the appendix.

Computing Prefix Probabilities

The problem of computing prefix probabilities for stochastic context-free languages has been discussed in [JL91, Sto95]. The main idea leading to the solution of this problem is that all parts of

[7]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.

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 [Sch92] (see Section 2.5.2).

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.

$$\frac{}{\epsilon \rightarrow^*_{ver} \epsilon} \tag{2.11}$$

$$\frac{A[\,] \rightarrow^*_{ver} v \quad \alpha \rightarrow^*_{ver} w \quad \alpha \neq \epsilon}{A[\,]\, \alpha \rightarrow^*_{ver} v\, w} \tag{2.12}$$

$$\frac{A[\,] \rightarrow^* a}{A[\,] \rightarrow^*_{ver} a} \tag{2.13}$$

$$\frac{\begin{array}{ll} A[\,] \rightarrow^* B[\sigma] & \alpha \rightarrow^*_{ver} v_\alpha \\ B[\circ\circ] \rightarrow \alpha\, C[p\circ\circ]\, \beta & \beta \rightarrow^*_{ver} v_\beta \\ C[\,] \rightarrow^* D[\,] & \gamma \rightarrow^*_{ver} v_\gamma \\ D[p\circ\circ] \rightarrow \gamma\, E[\circ\circ]\, \delta & \delta \rightarrow^*_{ver} v_\delta \\ E[\sigma] \Rightarrow^* X & v_\alpha v_\beta v_\gamma v_\delta \neq \epsilon \end{array}}{A[\,] \rightarrow^*_{ver} v_\alpha\, v_\gamma\, X\, v_\delta\, v_\beta} \tag{2.14}$$

$$\frac{\begin{array}{ll} A[\,] \rightarrow^* B[\sigma] & lab \in \{ver, hor\} \\ B[\circ\circ] \rightarrow \alpha\, C[p \circ\circ]\, \beta & \alpha \rightarrow^*_{ver} v_\alpha \\ C[\,] \rightarrow^*_{lab} v\, D[\sigma]\, w & \beta \rightarrow^*_{ver} v_\beta \\ D[\,] \rightarrow^* E[\,] & \gamma \rightarrow^*_{ver} v_\gamma \\ E[p \circ\circ] \rightarrow \gamma\, F[\circ\circ]\, \delta & \delta \rightarrow^*_{ver} v_\delta \\ F[\sigma] \Rightarrow^* X & v_\alpha v_\beta v_\gamma v_\delta \neq \epsilon \end{array}}{A[\,] \rightarrow^*_{ver} v_\alpha\; v\; v_\gamma\; X\; v_\delta\; w\;\; v_\beta} \tag{2.15}$$

$$\frac{\begin{array}{l} A[\,] \rightarrow^*_{ver} v\; B[\,]\; w \\ B[\,] \rightarrow^*_{lab} x\; C[\,]\; y \quad lab \in \{ver, hor\} \end{array}}{A[\,] \rightarrow^*_{hor} v\; x\; C[\,]\; y\; w} \tag{2.16}$$

$$\frac{\begin{array}{l} A[\,] \rightarrow^*_{ver} v\; B[\,]\; w \\ B[\,] \rightarrow^*_{ver} x \end{array}}{A[\,] \rightarrow^*_{ver} v\; x\; w} \tag{2.17}$$

$$\frac{\begin{array}{l} A[\,] \Rightarrow^* B[\sigma] \\ B[\,] \rightarrow^*_{hor} v\; C[\,]\; w \\ C[\sigma] \Rightarrow^* X \qquad \sigma \neq \epsilon \end{array}}{A[\,] \rightarrow^*_{ver} v\; X\; w} \tag{2.18}$$

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 2.14 and $A[\,] \rightarrow^* a$ in step 2.13 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 2.14 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 2.14 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.

A full treatment of computation of prefix probabilities for any input stochastic TAG is given in Appendix B (also see [NSS98]).

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 2.22. 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 2.14. From these two parts, the part labelled *c* is computed applying step 2.16. 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 2.18. Again vertically, step 2.15 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 2.22, can be partitioned in exactly one way according to the deduction steps.

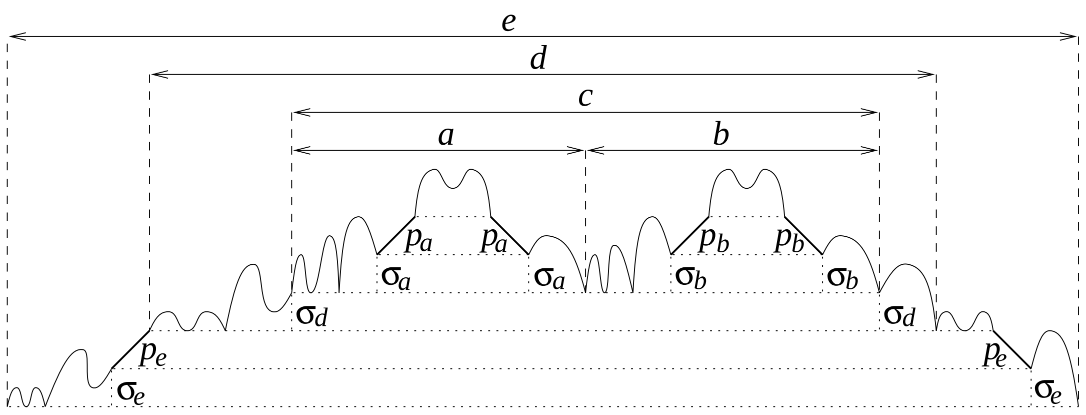


Figure 2.22: Development of the stack along a spine, and partitioning according to deduction steps.

2.6 Related Work

In this section we give an overview of some additional work done within the area of stochastic Tree Adjoining Grammars. We also compare the work in stochastic TAGs with work done in neighbouring areas like Data Oriented Parsing.

2.6.1 Work in Stochastic TAG and Related Areas

[Hwa98] uses the inside-outside algorithm for TAGs given in [Sch92] and applies it to stochastic Tree Insertion Grammars. The algorithm is combined with the use of bracketed Treebank data [PS92] as a source of a partial bracketing of the training sentences. The experiments reported were conducted on the WSJ Penn Treebank with the input to the learning algorithm being part-of-speech tags (rather than the words themselves). [Hwa99] extends the partial bracketing approach by suppressing various kinds of labeled brackets as a possible way of minimizing annotation cost by recovering some labeled brackets automatically.

[Chi01, Chi00] gives a statistical parser based on stochastic Tree Insertion Grammars. The experiments were based on fully lexicalized elementary trees and achieves 87.6% labeled precision and 87.4% labeled recall. These results show that one does not have to sacrifice performance over lexicalized PCFGs while maintaining a more elaborate model using TAGs. [Chi01] also reports results on the Chinese Treebank. This involved only minor changes to the English parser.

[XHPJ01, Xia01] reports on algorithms that permit the extraction of TAG derivation trees from Treebanks in various languages. The algorithms use only minimal edits to tables of data that are localized to each new Treebank.

[NSS98] shows that it is possible to extend the notion of assigning a probability to an entire sentence (defined in Section 2.4) to an arbitrary prefix of an input sentence. This extends a result shown for stochastic CFGs by [JL91, Sto95].

[SJ99, Sri97c, Sri97b] describes a method of partial parsing that uses local attachment heuristics after a probabilistic method that picks the best elementary tree for each word in a sentence: a technique termed as SuperTagging indicating the affinity between the problems of assigning complex structures such as trees to each word in a sentence as compared to the assignment of part of speech tags.

More distantly related use of elementary trees (of depth greater than 1) for statistical parsing occur in the works of [Sim00] and [SB98]. The work reported in [SB98] is related to the use of SuperTagging for partial parsing in the work of [Sri97b].

2.6.2 Comparison with Other Work

[Chi01] points out several similarities and differences between statistical parsing using TAGs and other tree-based statistical parsers like DOP [Bod01]. We briefly summarize some of these points here. The DOP1 model is related to stochastic TAGs since they both use probability measures over tree grammars. However, a stochastic TAG parser computes the best derivation tree for the input sentence, while DOP1 computes the best derived tree (or parse tree) by summing over all possible derivations that yield that tree. Here we have argued for the primacy of the derivation tree over the parse tree. Computing the best derivation provides an interface to future processing, for example, to compute a semantics for the input sentence while according to the TAG viewpoint, the derived tree provides no such benefit. Other benefits of using the DOP approach are the use of non-lexicalized trees and the use of trees with multiple lexical items at its leaf nodes. The use of both of these kinds of trees are already possible in the non-statistical uses of TAG (such as the parser used to parse the XTAG English Grammar [XTA01]). Extensions to stochastic TAGs and their use in statistical parsing have not yet shown improvements over the simpler form of stochastic TAG which is purely lexicalized and has only one lexical item per elementary tree.

[Hoo01] considers an extension of DOP with the operation of insertion taken from Tree Insertion Grammar. Hoogweg gives a compartive analysis of his experiments with DOP with and without the use of insertion. Hoogweg maintains the primacy of the parse tree where all possible derivations of a parse tree are computed, while in stochastic TAG and related work only the best derivation tree is ever computed based on a generative probability model. In linguistic terms, in an LTAG the elementary trees are semantically encapsulated objects and the derivation tree in LTAG leads to a compositional semantics. This is not a necessary requirement for DOP.

2.7 Conclusion

We have given an introduction to a formalism called Tree Adjoining Grammars (TAGs) that is useful in defining linguistic descriptions that are structurally complex. TAGs accomplish this by using trees, or even directed acyclic graphs, as elementary objects. Also in this chapter we have shown how these descriptions are useful in the context of statistical parsing. We provide a review of the definitions and results in the field of stochastic TAGs and show how the definition of stochastic TAG is well-defined. Finally, we have provided a brief comparison of stochastic TAGs and related work as well other tree based work in parsing such as DOP.

Chapter 3

Co-Training Methods for Statistical Parsing

In this chapter, we describe a novel Co-Training method for statistical parsing. The algorithm takes as input a small corpus (9695 sentences) annotated with parse trees, a dictionary of possible lexicalized structures for each word in the training set and a large pool of unlabeled text. The algorithm iteratively labels the entire data set with parse trees. Using empirical results based on parsing the Wall Street Journal corpus we show that training a statistical parser on the combined labeled and unlabeled data strongly outperforms training only on the labeled data. We also report on an experiment which performed Co-training on two statistical parsers which had different probability models. This experiment used the entire 1M word Penn Treebank as the labeled data and a 23M word WSJ corpus as the unlabeled set. The results were not as compelling as the first experiment and the chapter discusses the various reasons for this.

3.1 Introduction

The current crop of statistical parsers share a similar training methodology. They train from the Penn Treebank [MSM93]; a collection of 40,000 sentences that are labeled with corrected parse trees (approximately a million word tokens). In this chapter, we explore 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 9695 sentences of bracketed data (234467

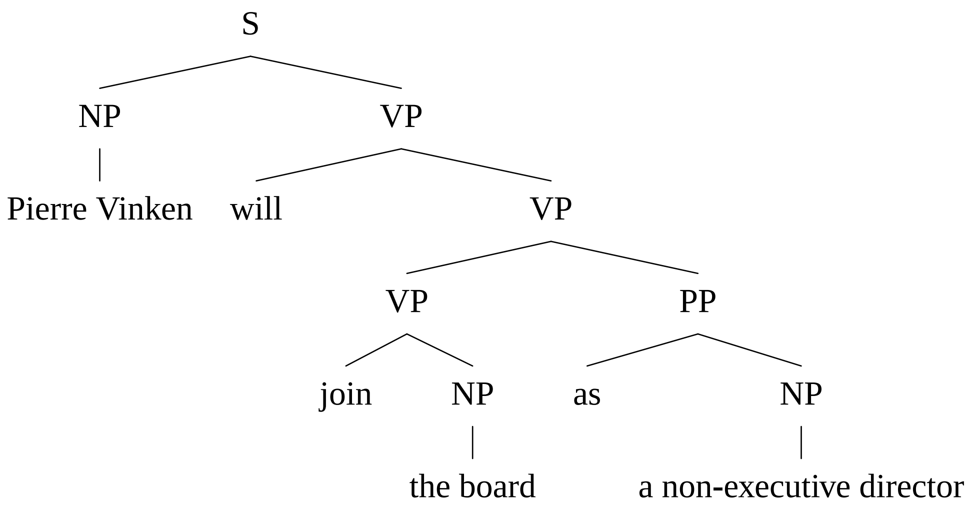


Figure 3.1: An example of the kind of output expected from a statistical parser. Find the best tree for a given sentence (using a generative model): $\arg\max_T P(T, S)$

word tokens). 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 [Yar95, BM98, GZ00] that has been used previously to train classifiers in applications like word-sense disambiguation [Yar95], document classification [BM98] and named-entity recognition [CS99] and apply this method to the more complex domain of statistical parsing.

3.2 Unsupervised techniques in language processing

While machine learning techniques that exploit annotated data have been very successful in attacking problems in NLP, there are still some aspects which are considered to be open issues:

- Adapting to new domains: training on one domain, testing (using) on another.
- Higher performance when using limited amounts of annotated data.
- Separating structural (robust) aspects of the problem from lexical (sparse) ones to improve performance on unseen data.

In the particular domain of statistical parsing there has been limited success in moving towards unsupervised machine learning techniques (see Section 3.8 for more discussion). A more promising approach is that of combining small amounts of seed labeled data with unlimited amounts of unlabeled data to bootstrap statistical parsers. In this chapter, we use one such machine learning technique: Co-Training, which has been used successfully in several classification tasks like web page classification, word sense disambiguation and named-entity recognition.

Early work in combining labeled and unlabeled data for NLP tasks was done in the area of unsupervised part of speech (POS) tagging. [CKPS92] reported very high results (96% on the Brown corpus) for unsupervised POS tagging using Hidden Markov Models (HMMs) by exploiting hand-built tag dictionaries and equivalence classes. Tag dictionaries are predefined assignments of all possible POS tags to words in the test data. This impressive result triggered several follow-up studies in which the effect of hand tuning the tag dictionary was quantified as a combination of labeled and unlabeled data. The experiments in [Mer94, Elw94] showed that only in very specific cases HMMs were effective in combining labeled and unlabeled data.

However, [Bri97] showed that aggressively using tag dictionaries extracted from labeled data could be used to bootstrap an unsupervised POS tagger with high accuracy (approx 95% on WSJ data). We exploit this approach of using tag dictionaries in our method as well (see Section 3.4.2 for more details). It is important to point out that, before attacking the problem of parsing using similar machine learning techniques, we face a representational problem which makes it difficult to define the notion of tag dictionary for a statistical parser.

The problem we face in parsing is more complex than assigning a small fixed set of labels to examples. If the parser is to be generally applicable, it has to produce a fairly complex "label" given an input sentence. For example, given the sentence *Pierre Vinken will join the board as a non-executive director*, the parser is expected to produce an output as shown in Figure 3.1.

Since the entire parse cannot be reasonably considered as a monolithic label, the usual method in parsing is to decompose the structure assigned in the following way:

S(join) $\rightarrow$ NP(Vinken) VP(join)

NP(Vinken) $\rightarrow$ Pierre Vinken

VP(join) $\rightarrow$ will VP(join)

VP(join) $\rightarrow$ join NP(board) PP(as)

. . .

However, such a recursive decomposition of structure does not allow a simple notion of a tag dictionary. We solve this problem by decomposing the structure in an approach that is different from that shown above which uses context-free rules.

The approach uses the notion of tree rewriting as defined in the Lexicalized Tree Adjoining Grammar (LTAG) formalism [JS92][1] which retains the notion of lexicalization that is crucial in the success of a statistical parser while permitting a simple definition of tag dictionary. For example, the parse in Figure 3.1 can be generated by assigning the structured labels shown in Figure 3.2 to each word in the sentence (for simplicity, we assume that the noun phrases are generated here as a single word). We use a tool described in [XPJ00] to convert the Penn Treebank into this representation.

Combining the trees together by rewriting nodes as trees (explained in Section 3.3) gives us the parse tree in Figure 3.1. A history of the bi-lexical dependencies that define the probability model used to construct the parse is shown in Figure 3.3. This history is called the *derivation tree*.

In addition, as a byproduct of this kind of representation we obtain more than the phrase structure of each sentence. We also produce a more embellished parse in which phenomena such as predicate-argument structure, subcategorization and movement are given a probabilistic treatment.

3.3 The Generative Model of the Statistical Parser

A stochastic LTAG derivation proceeds as follows [Sch92, Res92]. An initial tree is selected with probability P_i and other trees selected by words in the sentence are combined using the

[1]This is a lexicalized version of Tree Adjoining Grammar [JLT75, Jos85].

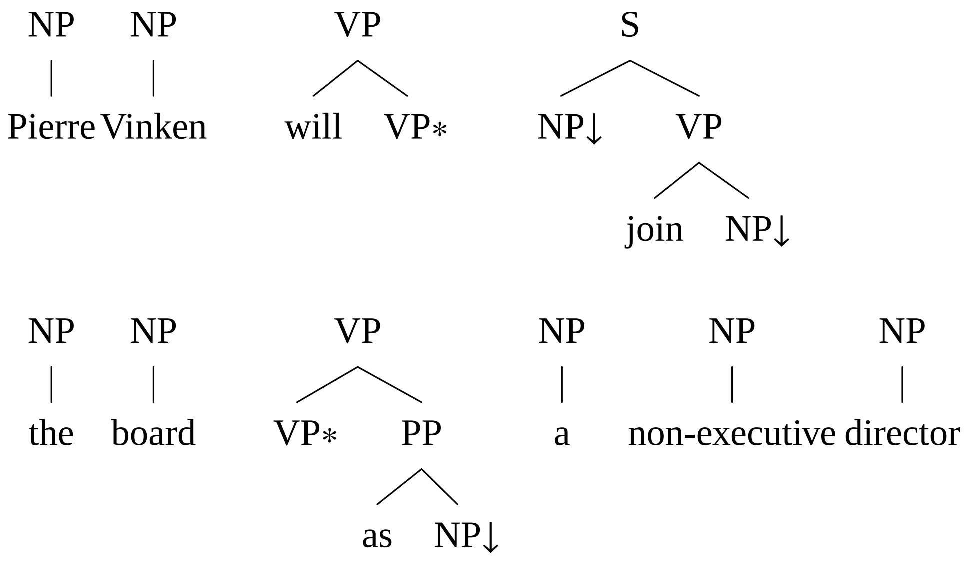


Figure 3.2: Parsing as tree classification and attachment.

operations of substitution and adjoining. These operations are explained below with examples. Each substutition is performed with probability P_S and each adjunction with probability P_a.

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

$$\sum_{\tau} P_i(\tau) = 1$$

Substitution is defined as rewriting a node η in the frontier of a tree t with probability P_S which is said to be proper if:

$$\sum_{\alpha} P_S(t, \eta \rightarrow \alpha) = 1$$

where $t, \eta \rightarrow \tau'$ indicates that tree α is substituting into node η in tree t. An example of the operation of substitution is shown in Figure 3.4.

Adjoining is defined as rewriting any internal node η of a tree t by another tree β. This is a recursive rule and each adjoining operation is performed with probability P_a which is proper if:

$$P_a(t, \eta \rightarrow \textsc{na}) + \sum_{\beta} P_a(t, \eta \rightarrow \beta) = 1$$

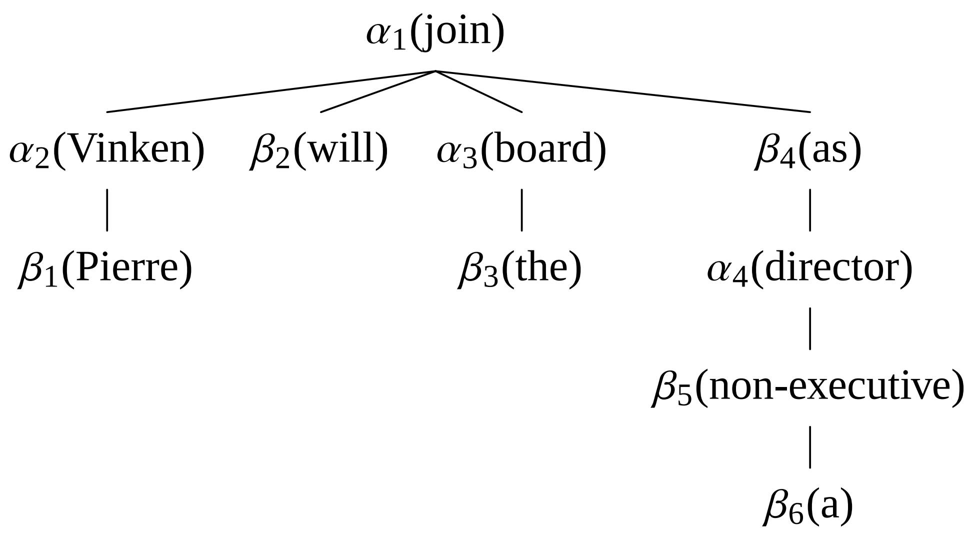


Figure 3.3: A derivation tree indicating all the attachments between trees based on the most likely bi-lexical dependencies that have occurred during the parse of the sentence.

P_a here is the probability that β rewrites an internal node η in tree t or that no adjoining (NA) occurs at node η in t. The additional factor that accounts for no adjoining at a node is required for the probability to be well-formed. An example of the operation of adjoining is shown in Figure 3.5.

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

$$\begin{aligned} Pr(\mathcal{D}, w_0 \dots w_n) = \\ & P_i(\alpha, w_i) \times \prod_p P_s(\tau, \eta, w \rightarrow \alpha, w') \times \\ & \prod_q P_a(\tau, \eta, w \rightarrow \beta, w') \times \prod_r P_a(\tau, \eta, w \rightarrow \textsc{na}) \end{aligned}$$

In the next section we show how to exploit this notion of tag dictionary to the problem of statistical parsing.

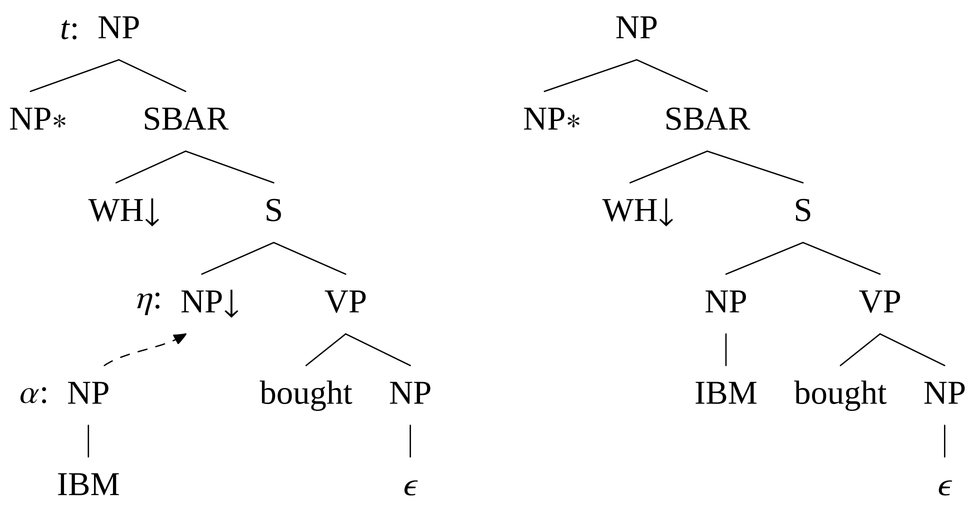


Figure 3.4: Example substitution: $\sum_\alpha P_S(t, \eta \rightarrow \alpha) = 1$

3.4 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 and large set of unlabeled data.
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.

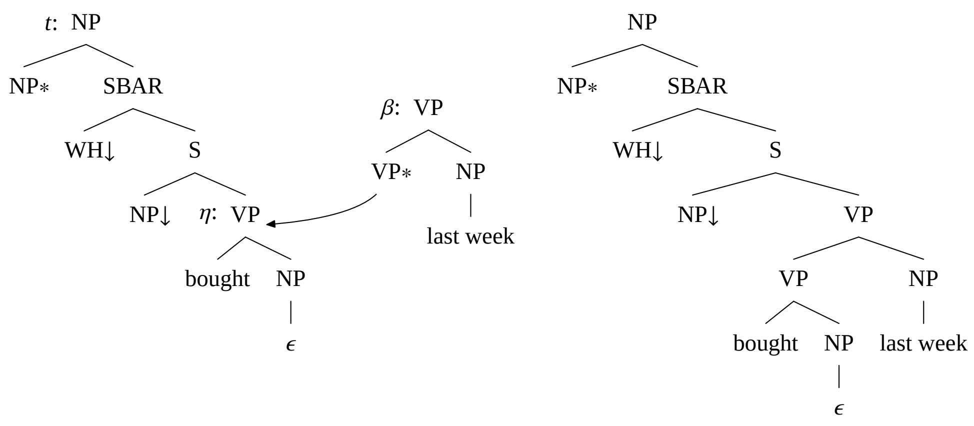


Figure 3.5: Example adjunction: $\mathcal{P}_a(t, \eta \rightarrow \textsc{na}) + \sum_\beta \mathcal{P}_a(t, \eta \rightarrow \beta) = 1$

Co-Training has been used before in applications like word-sense disambiguation [Yar95], web-page classification [BM98] and named-entity identification [CS99]. 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 2–3), 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 3.8.

Co-training [BM98, Yar95] can be informally described in the following manner:

- Pick two (or more) "views" of a classification problem.
- Build separate models for each of these "views" and train each model on a small set of labeled data.
- Sample an unlabeled data set and to find examples that each model independently labels with high confidence. [NG00]
- Confidently labeled examples can be picked in various ways. [CS99, GZ00]
- Take these examples as being valuable as training examples and iterate this procedure until the unlabeled data is exhausted.

Effectively, by picking confidently labeled data from each model to add to the training data, *one model is labeling data for the other model.*

3.4.1 Lexicalized Grammars and Mutual Constraints

In the representation we use, 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 3.2).

2. Finding the correct attachments between these structures to get the best parse (as shown in Figure 3.1).

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 predicted by the second model[2]. This represents the mutual constraint that each model places on the other.

3.4.2 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 training data. This information is stored in a

[2]See §3.8 for a discussion of the relation of this approach to that of SuperTagging [Sri97a]

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.

We use a count cutoff for trees in the labeled data and combine observed counts into an *unobserved* tree count. This is similar to the usual technique of assigning the token *unknown* to infrequent word tokens. In this way, trees unseen in the labeled data but in the tag dictionary are assigned a probability in the parser.

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 [Bri97] where seed data of which POS tags can be selected by each word is given as input to the unsupervised tagger.

After initial experiments reported later in this chapter, We went back and re-evaluated the importance of the tag dictionaries to the performance of the method. We added tag dictionaries with previously unseen trees smoothed with a small held-over probability that was obtained by omitting those trees that occurred only once in the small labeled set. While the error rate decreased by 10%, this improvement was much lower than the performance increase obtained by applying the co-training approach described in this chapter.

3.5 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)

3.5.1 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 [Sri97a]. 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 [Sri97a]).

We conducted several experiments in using *n*-best SuperTagging and combining the subsequent set of trees to form a parse. Detailed descriptions of these experiments are given in Chapter 6 in Section 6.2.2. Careful consideration was given to the problem that SuperTagging does not always provide a set of trees that can be stapled together to form a parse. In particular, the value of *n* was set high enough and the set of elementary trees were carefully chosen (by setting the appropriate parameters in the grammar extraction module which gives elementary trees from the Treebank) to make a full parse always possible.

$$
\begin{aligned}
&P(\mathbf{T}|\mathbf{W}) \\
&\quad = P(T_0 \dots T_n | W_0 \dots W_n) && (3.1)\\
&\quad = \frac{P(T_0 \dots T_n) \times P(W_0 \dots W_n | T_0 \dots T_n)}{P(W_0 \dots W_n)} && (3.2)\\
&\quad \approx P(T_i | T_{i-2} T_{i-1}) \times P(W_i | T_i) && (3.3)
\end{aligned}
$$

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

We get (3.2) by using Bayes theorem and we obtain (3.3) from (3.2) 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.

3.5.2 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_i (introduced earlier in 3.3) stand for the probability of being root of a derivation tree defined as follows:

$$\sum_{\tau} P_i(\tau) = 1$$

including lexical information, this is written as:

$$\Pr(\tau, w, p | top = 1) =$$

$$\Pr(\tau|top = 1) \times \tag{3.4}$$

$$\Pr(p|\tau, top = 1) \times \tag{3.5}$$

$$\Pr(w|\tau, p, top = 1); \tag{3.6}$$

where the variable top indicates that τ is the tree that begins the current derivation. There is a useful approximation for P_i:

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

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

$$\hat{\Pr}(label|top = 1) = \frac{Count(top = 1, label) + \alpha}{Count(top = 1) + N\alpha} \tag{3.7}$$

where N is the number of bracketing labels and α is a constant used to smooth zero counts.

Let P_{ATTACH} stand for either P_S or P_a (introduced earlier in 3.3). Thus, P_{ATTACH} stands for the probability of either substituting or adjoining τ' into another τ:

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

including lexical information, this is written as:

$$\Pr(\tau', p', w'|Node, \tau, w, p) \tag{3.8}$$

$$\Pr(\text{NA}|Node, \tau, w, p) \tag{3.9}$$

We decompose (3.8) into the following components:

$$\Pr(\tau', p', w'|Node, \tau, w, p) =$$

$$\Pr(\tau'|Node, \tau, w, p) \times \tag{3.10}$$

$$\Pr(p'|\tau', Node, \tau, w, p) \times \tag{3.11}$$

$$\Pr(w'|p', \tau', Node, \tau, w, p); \tag{3.12}$$

We do a similar decomposition for (3.9).

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_i and P_{ATTACH}, let $c = Count(e_1)$. 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 D is the diversity of e_1 (i.e. the number of distinct counts for e_1).

For P_{ATTACH} we further smooth probabilities (3.10), (3.11) and (3.12). We use (3.10) as an example, the other two are handled in the same way.

$$\hat{\Pr}(\tau' | Node, \tau, w, p) = \frac{(Count(Node, \tau, w, p, \tau') + \alpha)}{(Count(Node, \tau, w, p) + k\alpha)} \tag{3.13}$$

$$Count(Node, \tau, w, p) = \sum_{y \in \mathcal{T}'} Count(Node, \tau, w, p, y) \tag{3.14}$$

where k is the diversity of adjunction, that is: the number of different trees that can attach at that node. $\mathcal{T}'$ is the set of all trees τ' that can possibly attach at Node in tree τ.

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

The parsing algorithm that is used to decode a parse for an input sentence is given in Chapter 6.

3.6 Co-Training algorithm

We are now in the position to describe the Co-Training algorithm, which combines the models described in Section 3.5.1 and in Section 3.5.2 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 3.4.2.

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* and *unlabeled*
2. Update cache
   - Randomly select sentences from *unlabeled* and refill *cache*
   - If *cache* is empty; exit
3. Train models H1 and H2 using *labeled*
4. Apply H1 and H2 to cache.
5. Pick most probable n from H1 (stapled together) and add to *labeled*.
6. Pick most probable n from H2 and add to *labeled*
7. $n = n + k$; Go to Step 2

For the experiment reported here, $n = 10$, and k was set to be n in each iteration. We ran the algorithm for 12 iterations (covering 20480 of the sentences in *unlabeled*) and then added the best parses for all the remaining sentences.

3.7 Experiment

3.7.1 Setup

The experiments we report were done on the Penn Treebank WSJ Corpus [MSM93]. The various settings for the Co-Training algorithm (from Section 3.6) are as follows:

- *labeled* was set to Sections 02-06 of the Penn Treebank WSJ (9625 sentences)
- *unlabeled* was 30137 sentences (Section 07-21 of the Treebank stripped of all annotations).
- A tag dictionary of all lexicalized trees from *labeled* and *unlabeled*.
- Novel trees were treated as unknown tree tokens.
- The *cache* size was 3000 sentences.

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 allows 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 because shorter sentences 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.

3.7.2 Results

We scored the output of the parser on Section 23 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.

We used EVALB (written by Satoshi Sekine and Michael Collins) which scores based on PARSEVAL [BAF$^+$91]; with the standard parameter file (as per standard practice, part of speech brackets were not part of the evaluation). Also, we used Adwait Ratnaparkhi's part-of-speech tagger [Rat96] to tag unknown words in the test data.

We obtained 80.02% and 79.64% labeled bracketing precision and recall respectively (as defined in [BAF$^+$91]). The baseline model which was only trained on the 9695 sentences of labeled data performed at 72.23% and 69.12% precision and recall. These results show that training a statistical parser using our Co-training method to combine labeled and unlabeled data strongly outperforms training only on the labeled data.

It is important to note that unlike previous studies, our method of moving towards unsupervised parsing are directly compared to the output of supervised parsers.

Certain differences in the applicability of the usual methods of smoothing to our parser cause the lower accuracy as compared to other state of the art statistical parsers. However, we have consistently seen increase in performance when using the Co-Training method over the baseline across several trials. It should be emphasised that this is a result based on less than 20% of data that is usually used by other parsers.

3.8 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 [Sri97a] 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 [LY90, CR98a]. However, there are several limitations of the inside-outside algorithm for unsupervised parsing, see [Mar95] 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 [LST92, FW96]. However, there are still inherent computational limitations due to the vast search space (see [PPG+94] for discussion). None of these approaches can even be realistically compared to supervised parsers that are trained and tested on the kind of representations and the complexity of sentences that are found in the Penn Treebank.

[CJ98] combine unlabeled and labeled data for parsing with a view towards language modeling applications. The goal in their work 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 [Yar95, BM98, GZ00, CS99]. [Yar95] 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 segment of 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. [BM98] further embellish 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. [CS99] further extend the use of classifiers that have mutual constraints by adding terms to AdaBoost which force the classifiers to agree (called Co-Boosting). [GZ00] 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.

3.9 Further Experiments with Larger Sets of Labeled Data

We conducted some further experiments in which the role of the tag dictionary was de-emphasized by training on the entire Treebank but using a separate set of 23M words of WSJ data as the unlabeled set.

In addition, we also explored the attractive notion of performing co-training between two statistical parsers, each parser having conditionally independent lexicalized models thus learning different features from the labeled set.

The two parsing models we used were both TAG-based but had strikingly different lexicalized probability models. One was based on the traditional notion of adjuction in TAG: only one adjunction per node was permitted in this model. This caused the stacking of modifiers one on another in the lexicalized probability model. For example, in the phrase *saw a man with a telescope using a tripod*, the PP headed by *using* adjoins onto the root node of the PP headed by *with* since only one adjunction is possible at the VP node of the tree headed by *saw* giving us the lexical dependencies of *with, saw* and *using, with*.

The second model was based on the Tree Insertion Grammar model (see introduction in Chapter 2. In this model, multiple adjunction is permitted at each node and wrapping trees produced by one left auxiliary tree adjoining onto the root node of a right auxiliary tree (or vice versa) is disallowed. As a result, modifiers all modify the head together thus changing the lexicalized probability model. For example, in the phrase *saw with a telescope using a tripod*, the PP headed by *using* and the PP headed by *with* both adjoin onto the VP node of the tree headed by *saw* giving us the lexical dependencies of *with, saw* and *using, saw*.

We trained parsers using each of these models on the standard training set of sections 02-21 of the Penn Treebank. We then performed co-training using a larger set of WSJ unlabeled text (23M words). Even after 12 iterations of co-training, performance did not improve significantly over the baseline of LR 85.2% and LP 86%.

There are three possible reasons why this experiment did not succeed. We discuss them below and give reasons why one of these reasons looks the most likely.

- There was no substantial overlap between the features used in each of the probability models. This reason is surprising because we explicitly picked the two models to be as different as possible from each other. However, after training the two models we compared the distinctly different lexicalized features between them. We found that only 22% of the features were different. Most of the lexicalized features were found in either model but with different probabilities. Due to this we believe that this is the most likely reason why the experiment did not go as planned.

- There was enough labeled data and further addition of unlabeled data will not improve performance. There is evidence from the study in [Gil01] that the lexicalized features (those that involve bigrams of lexical items) are under-utilized when parsing the test data. For this reason, we think that unlabeled data if correctly used can improve performance in a co-training setting.

- The tag dictionary that was used in the first experiment was crucial in the success there. Since we used unlabeled data for which no such dictionary existed, the lack of such a tag dictionary was the reason why this experiment using a larger labeled and unlabeled set did not succeed. While the improvement in the first experiment was more substantial due to the additional information in the tag dictionary, in this second experiment the lack of tag dictionary does not seem to be the critical factor. The reason for this is that the lack of an appropriate tag dictionary causes parsing failure because a crucial tree was missing. In this second experiment, the elementary trees were considered to be a closed set and unknown words were assigned trees based on their part-of-speech tags (we tagged the unlabeled set using Adwait's tagger). We did not find an unusually high number of failures which might suggest a problem with the tag dictionary.

We expect to continue with experiments involving Co-training based on larger labeled and unlabeled sets. The two avenues we wish to pursue are (1) to use a partial parser like we did in our first experiment, and (2) to use two parsers but use bagging to introduce variation in the lexicalized features used by the parser. In addition, we plan to pursue the co-training of multiple parsers at the constituent level during the Johns Hopkins summer workshop in 2002.

3.10 Co-training and the EM algorithm

In this section we explore the relationship between the Co-training algorithm and the EM algorithm. The EM algorithm is a theoretically well-justified method for the recovery of parameter values in a generative model under maximum likelihood assumptions for hidden data. A common instantiation of the EM algorithm for parameter estimation in context-free and TAG is the inside-outside algorithm (see Chapter 2 for a definition of the IO algorithm for TAGs).

[NG00] is a study which tries to separate two factors in the family of Co-training algorithms versus EM-based algorithms:

1. The gradient descent aspect of EM vs. the iterative nature of co-training, and

2. The generative model used in EM vs. the conditional independence between the features used by the two models that is exploited in co-training.

Figure 3.10 is graphical depiction of this dichotomy between Co-training and EM and shows both the similarity and differences between the two approaches to the use of unlabeled data. In the figure, $*$ refers to the work in [NG00] and † refers to work in discriminative objective functions $f; Q_0 \parallel Q_{dis}$ by Tom Mitchell (personal communication). In this comparision, $Q_0 \parallel Q_\infty$ is the Kullback-Liebler distance between the data distribution Q_0 and the distribution of the data given by the convergent setting of the parameter values in the model given by Q_∞. See [Hin00] for a different approach to the use of ensemble learning and its relation to EM.

In Figure 3.10, we point out that one could construct a discriminative objective function Q_{dis} over which iterative EM-based re-estimation or gradient descent can be done to take advantage of the conditional independence of features used in two (or more) models trained on the labeled data. Here is one way to define such an objective function. Let the labeled data consist of pairs x, y where x is labeled as y and the learner has to learn to map unseen x items to labels y. Let us take each x to be the combination of two features $\langle x_1, x_2 \rangle$. We use two classifiers g_1 and g_2 where g_1 learns the mapping x_1, y while g_2 learns the mapping x_2, y. Let $g_1(x_i) = g_2(x_i) = y_i$ for i in the labeled data. Then the objective function can be defined as:

$$\sum_i (\hat{g}_1(x_i) - y_i)^2 + (\hat{g}_2(x_i) + y_i)^2 + (P(y_i) \mid_{\text{lab}} = P(y_i) \mid_{\text{unlab}})$$

The last term in the equation keeps the marginal distribution of the labels stable even with the addition of new data taken from the unlabeled set. This function can provide the basis for re-estimation over the labeled and unlabeled set. The nature of this re-estimation has to be carefully controlled. For example, EM has been used successfully in text classification in combination of labeled and unlabeled data (see [NMTM99]). However, the use of EM in this case required some central assumptions about the use of unlabeled data (see [Nig01] for further details).

The most important distinction between Co-training and EM is the greedy nature of Co-training and the use of sampling to avoid the exhaustive search of the joint distribution that is done in EM-based algorithms like Inside-Outside. The time taken by Co-training in learning from unlabeled data is thus not as exhorbitant as the IO algorithm.

	max likelihood over full unlabeled set	iterative selection from unlabeled set
$Q_0 \parallel Q_\infty$	EM†	self-training
conditionally independent features	co-EM*	Co-Training

Figure 3.6: Comparison of Co-training with the EM algorithm.

3.11 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 of 9695 sentences (234467 word tokens) of bracketed data, a large pool of unlabeled text and a tag dictionary of lexicalized structures for each word in this training set (based on the LTAG formalism). The algorithm presented iteratively labels the unlabeled data set with parse trees. We then train a statistical parser on the combined set of labeled and unlabeled data.

We obtained 80.02% and 79.64% labeled bracketing precision and recall respectively. The baseline model which was only trained on the 9695 sentences of labeled data performed at 72.23% and 69.12% precision and recall. These results show that training a statistical parser using our Co-training method to combine labeled and unlabeled data strongly outperforms training only on the labeled data.

It is important to note that unlike previous studies, our method of moving towards unsupervised parsing can be directly compared to the output of supervised parsers. Unlike previous approaches to unsupervised parsing our method can be trained and tested on the kind of representations and the complexity of sentences that are found in the Penn Treebank.

In addition, as a byproduct of our representation we obtain more than the phrase structure of each sentence. We also produce a more embellished parse in which phenomena such as predicate-argument structure, subcategorization and movement are given a probabilistic treatment.

Chapter 4

Learning Unknown Subcategorization Frames

In this chapter, we present some novel machine learning techniques for the identification of subcategorization information for verbs in Czech. We compare three different statistical techniques applied to this problem. We show how the learning algorithm can be used to discover previously unknown subcategorization frames from the Czech Prague Dependency Treebank. The algorithm can then be used to label dependents of a verb in the Czech treebank as either arguments or adjuncts. Using our techniques, we are able to achieve 88% precision on unseen parsed text.

4.1 Introduction

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. [CM98, CR98b] 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 [Man93, BC97]. 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 [WM89, Bre91, Bre93, Bre94, UEGW93, Man93, EC96, BC97, CM98, CR98b].

All of this work deals with English. In this chapter we report on techniques that automatically extract SFs for Czech, which is a free word-order language, where verb complements have visible case marking.

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 [Haj98]) where the subcategorization information is *not* given; although this might be considered a simpler problem as compared 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.

4.2 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.

4.2.1 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 chapter sidesteps this issue altogether, since it is left to the algorithm to learn what is an appropriate SF for a verb.

Figure 4.1 shows a sample input sentence from the PDT annotated with dependencies which is used as training material for the techniques described in this chapter. 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[1]. 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 4.1 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.

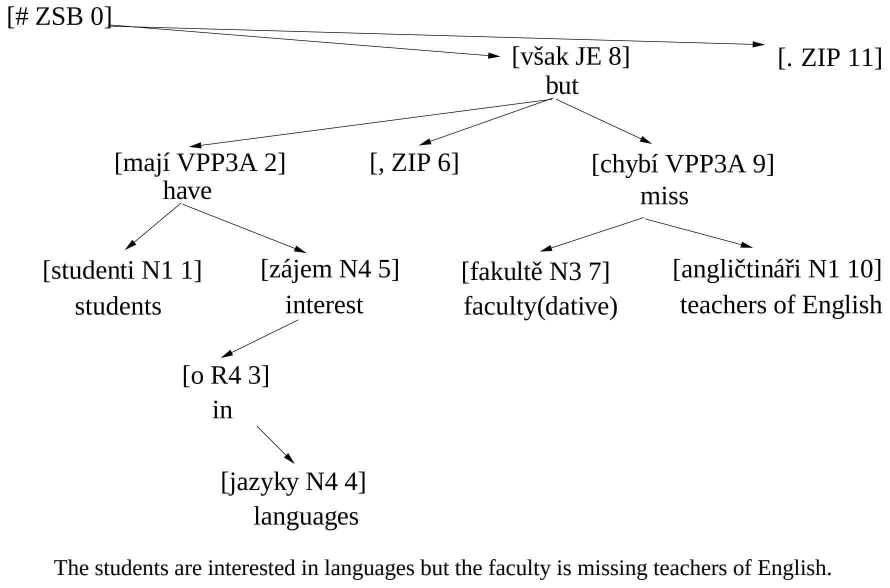


Figure 4.1: Example input to the algorithm from the Prague Dependency Treebank

[1]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.

4.2.2 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 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 otv´ı r´a soubor. (SVO: Martin opens the file)
b. Soubor otv´ı r´a Martin. (OVS: ≠ the file opens Martin)
c. Martin soubor otv´ı r´a.
d. #Otv´ı r´a Martin soubor.
e. #Otv´ı r´a soubor Martin.
f. Soubor otv´ı r´a 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 4.2 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.

4.2.3 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 [HH98, Haj98]; 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 4.2.1).

4.3 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 4.2.1), 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 4.3.3.

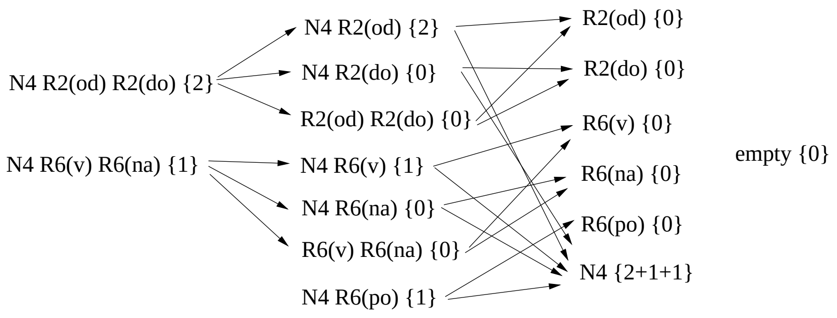


Figure 4.2: Computing the subsets of observed frames for the verb *absolvovat*. The counts for each frame are given within braces {}. In this example, the frames *N4 R2(od), N4 R6(v)* and *N4 R6(po)* have been observed with other verbs in the corpus. Note that the counts in this figure do not correspond 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 [BD77, Dun93].

4.3.1 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 [BD77](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 [Dun93] is given by:

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

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.

4.3.2 T-scores

Another statistic that has been used for hypothesis testing is the *t-score*. Using the definitions from Section 4.3.1 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.

$$T = \frac{p_1 - p_2}{\sqrt{\frac{p_1(1-p1)}{n1} + \frac{p_2(1-p2)}{n2}}}$$

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.

4.3.3 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 papers on SF extraction starting with [Bre91, Bre93, Bre94].

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.

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

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 4.2 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.

4.4 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 4.2.1 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 chapter is shown in Table 4.1. 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.

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.

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).

	Baseline 1	Baseline 2	Likelihood Ratio	T-scores	Hypothesis 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
Known verb nodes	1027	981	981	981	907
Known verb complements	2144	2010	2010	2010	1812
Correct Suggestions	1187.5	1573.5	1642.5	1652.9	1596.5
True Args	956.5	910.5	910.5	910.5	834.5
Suggested Args	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 4.1: Comparison between the baseline methods and the proposed methods.

4.5 Comparison with related work

Preliminary work on SF extraction from corpora was done by [Bre91, Bre93, Bre94] and [WM89, UEGW93]. Brent [Bre93, Bre94] uses the standard method of testing miscue probabilities for filtering frames observed with a verb. [Bre94] presents a method for estimating p_f. Brent applied his method to a small number of verbs and associated SF types. [Man93] applies Brent's method to parsed data and obtains a subcategorization dictionary for a larger set of verbs. [BC97, CM98] differs from earlier work in that a substantially larger set of SF types are considered; [CR98b] 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. [BV98] 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 4.2.

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 [BC97, CM98], 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 chapter) 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 4.3.3.

4.6 Conclusion

We are currently incorporating the SF information produced by the methods described in this chapter 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

Previous work	Data	#SFs	#verbs tested	Method	Miscue rate	Corpus
[UEGW93]	POS + FS rules	6	33	heuristics	NA	WSJ (300K)
[Bre93]	raw + FS rules	6	193	Hypothesis testing	iterative estimation	Brown (1.1M)
[Man93]	POS + FS rules	19	3104	Hypothesis testing	hand	NYT (4.1M)
[Bre94]	raw + heuristics	12	126	Hypothesis testing	non-iter estimation	CHILDES (32K)
[EC96]	Full parsing	16	30	Hypothesis testing	hand	WSJ (36M)
[BC97]	Full parsing	160	14	Hypothesis testing	Dictionary estimation	various (70K)
[CR98b]	Unlabeled	9+	3	Inside-outside	NA	BNC (5-30M)
Current	Fully Parsed	Learned 137	914	Subsets+ Hyp. testing	Estimate	PDT (300K)

Table 4.2: Comparison with previous work on automatic SF extraction from corpora

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 [SM99, LB99, Lap99, SMKW99a].

The statistical models in this chapter 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).

Chapter 5

Learning Verb Alternations from Corpora

In this chapter we investigate the task of automatically identifying the correct argument structure for a set of verbs. The argument structure of a verb allows us to predict the relationship between the syntactic arguments of a verb and their role in the underlying lexical semantics of the verb. Following the method described in [MS01], we exploit the distributions of some selected features from the local context of a verb. These features were extracted from a 23M word WSJ corpus based on part-of-speech tags and phrasal chunks alone. This annotation was minimal as compared to previous work on this task which used automatically parsed data. We constructed several decision tree classifiers trained on this data. The best performing classifier achieved an error rate of 33.4%. Our result compares very favorably with previous work despite using considerably less data and requiring only minimal annotation of the data.

5.1 Introduction

In this chapter 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. This is often termed as the classification of verb diathesis roles or the lexical semantics of predicates in natural language (see [Lev93, WM89, MK98, SM99, SMKW99b, Lap99, LB99, Sch00]).

Following the method described in [MS01, SM99, SMKW99b], we exploit the distributions of some selected features from the local context of a verb but we differ from these previous studies in the use of 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 such as noun-phrase and verb-phrase chunks to identify potential arguments of each verb.

Lexical knowledge acquisition plays an important role in corpus-based NLP. Knowledge of verb selectional preferences and verb subcategorization frames (SFs) can be extracted from corpora for use in various NLP tasks. However, knowledge of SFs is often not fine-grained enough to distinguish various verbs and the kinds of arguments that they can select. We consider a difficult task in lexical knowledge acquisition: that of finding the underlying argument structure which can be used to relate the observed list of SFs of a particular verb. The task involves identifying the roles assigned by the verb to its arguments. Consider the following verbs (the examples are taken from [MS01], each occuring with intransitive and transitive SFs[1].

Unergative

(2) a. The horse raced past the barn.
b. The jockey raced the horse past the barn.

Unaccusative

(3) a. The butter melted in the pan.
b. The cook melted the butter in the pan.

Object-Drop

(4) a. The boy washed.
b. The boy washed the hall.

Each of the verbs above occurs with both the intransitive and transitive SFs. However, the verbs

[1]See [Lev93] for more information. The particular categorization that we use here is motivated in [SM97]

differ in their underlying argument structure. Each verb assigns a different role to their arguments in the two subcategorization possibilities. For each verb above, the following lists the roles assigned to each of the noun phrase arguments in the SFs permitted for the verb. This information can be used for extracting appropriate information about the relationships between the verb and its arguments.

Unergative

INTRAN: NP_{agent} raced

TRAN: NP_{causer} raced NP_{agent}

Unaccusative

INTRAN: NP_{theme} melted

TRAN: NP_{causer} melted NP_{theme}

Object-Drop

INTRAN: NP_{agent} washed

TRAN: NP_{agent} washed NP_{theme}

Our task is to identify the transitive and intransitive usage of a particular verb as being related via this notion of argument structure. This is called the *argument structure classification* of the verb. In the remainder of this chapter we will look at the problem of placing verbs into such classes automatically.

Our results in this chapter serve as a replication and extension of the results in [MS01]. Our main contribution in this chapter is to show that with reasonable accuracy, this task can be accomplished using only tagged and chunked data. In addition, we incorporate some additional features such as part-of-speech tags and the use of subcategorization frame learning as part of our classification algorithm. Our result compares very favorably with previous work despite using considerably less data and requiring only minimal annotation of the data ([MS01] use a 65M word fully parsed WSJ corpus).

5.2 The Hypothesis

We create a probabilistic classifier that can automatically classify a set of verbs into argument structure classes with a reasonable error rate. We use the hypothesis introduced by [SM99] 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 work the additional hypothesis we wish to test is whether the application of subcategorization frame (SF) learning to this kind of learning technique will permit the use of noisy data with less annotation (chunked data without explicit SF information vs. automatically parsed text where SF information is known).

In the previous section we saw that we sometimes have noun-phrase arguments (NP_{causer}) as being a *causer* of the action denoted by the verb. For example, [SM99] show that a classifier can exploit these causativity facts to improve classifiction.

We use some new features in addition to the ones proposed and used in [MS01] for this task. In addition, we include as a feature the probabilistic classification of the verb as a transitive or intransitive verb. Thus the classifier is simulaneously placing each verb into the appropriate subcategorization frame as well as identifying the underlying thematic roles of the verb arguments.

In our experiment, we will consider the following set of classes (each of these were explained in the previous section): unergative, unaccusative, and object-drop. We test 76 verbs taken from [Lev93] that are in one of these three classes. The particular verbs were chosen to include high frequency as well as low frequency verb tokens in our particular corpus of 23M words of WSJ text.[2] We used all instances of these verbs from the WSJ corpus. The data was annotated with the right classification for each verb and the classifier was run on 10% of the data using 10-fold cross

[2]The particular verbs selected were looked up in [Lev93] and the class for each verb in the classification system defined in [SM97] was selected with some discussion with linguists.

validation. We describe the experiment in greater detail in Section 5.4.

5.3 Identifying subcategorization frames

An important part of identifying the argument structure of the verb is to find the verb's subcategorization frame (SF). For this chapter, we are interested in whether the verb takes an intransitive SF or a transitive SF.

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 chapter sidesteps this issue altogether, since it is left to the algorithm to learn what is an appropriate SF for a verb.

The problem of SF identification using statistical methods has had a rich discussion in the literature [UEGW93, CR98b, EC96, BV98, Man93, BC97, Bre93, Bre94]. In this chapter, we use the method of hypothesis testing to discover the SF for a given verb [Bre94]. Along with the techniques given in these papers, [SZ00, KGM00] also discuss other methods for hypothesis testing such the use of the t-score statistic and the likelihood ratio test. After experimenting with all three of these methods we selected the likelihood ratio test because it performed with higher accuracy on a small set of hand-annotated instances. We use the determination of the verb's SF as an input to our argument structure classifier (see Section 5.4).

The method works as follows: 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 by using the method of hypothesis testing using the likelihood ratio test. For further background on this method of hypothesis testing the reader is referred to [BD77, Dun93].

5.3.1 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 [BD77](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 [Dun93] is given by:

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

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. If this value is above a certain threshold it is taken to be a positive value for the binary feature TRAN, else it is a positive feature for the binary feature INTRAN in the construction of the classifier.

5.4 Steps in Constructing the Classifi er

To construct the classifier, we will identify features that can be used to accurately distinguish verbs into different classes. The features are computed to be the probability of observing a particular feature with each verb to be classified. We use C5.0 [Qui92] to generate the decision tree classifier. 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 [Rat96] and then running Steve Abney's chunker [Abn97] 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. The first four features were discussed and motivated in [SM99, MS01]. In some cases, we have modified the features to include information about part-of-speech tags. The discussion below clarifies the similarities and changes. The features we used in addition are the last two in the following list, the part-of-speech features and the subcategorization frame features. [3]

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

2. active (ACT) and passive (PASS)

3. causative (CAUS)

[3]Note that while [SM99, MS01] used a TRAN/INTRAN feature, in their case it was estimated in a completely different way using tagged data. Hence, while we use the same name for the feature here, it is not the same kind of feature as the one used in the cited work.

4. animacy (ANIM)
5. Part of Speech of the subject noun-phrase and object noun-phrase
6. transitive (TRAN) and intransitive (INTRAN)

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

5.4.1 Finding the main verb of the sentences

To find the main verb, we constructed a deterministic finite-state automaton that finds the main verb within the verb phrase chunks. This DFA is used in two steps. First, to select a set of main verbs from which we select the final set of 76 verbs used in our experiment. Secondly, the actual set of verbs is incorporated into the DFA in the feature selection step.

5.4.2 Obtaining the frequency distribution of the features

The general form of the equation we use to find the frequency distribution of each feature of the verb is the following:

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

where $P(V_j)$ is the distribution of feature j of the verb, N is the total number of features of the particular type (e.g., the total number of CAUS features or ANIM features as described below) 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.

5.4.3 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 chapter, we approximate the value by using the following approach. Also, the causative value is not a probability but a weight which is subsequently normalized.

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

$$\text{CAUS} = \frac{\text{overlap}}{\text{sum of all subject and objects in multiset}}$$

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 $\frac{(2+1)}{(4+3)} = \frac{3}{7}$.

Note that using this measure, we expect to get higher weights for tokens that occur frequently in the object position and sometimes in the subject position. For example, $\text{CAUS}(\{a, b\}, \{a, b\}) = \frac{2}{4}$ while $\text{CAUS}(\{a, b\}, \{a, a, a\}) = \frac{3}{5}$. This difference in the weight given by the CAUS feature is exploited in the classifier.

5.4.4 The animate feature: ANIM

Similar to CAUS, we can only approximate the value of animacy. We use 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*. In addition we use the set of part-of-speech tags which are associated with animacy in Penn Treebank tagset as part of set of features described in the next section.

5.4.5 Part of Speech of object and subject

The part-of-speech feature picks up several subtle cues about the differences in the types of arguments selected by the verb in its subject or object position.

We count the occurrence of the head nouns of the subject noun phrase and the 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, N is the total number of relevant POS features 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.

5.4.6 Transitive and intransitive SF of the verb

To find values for this feature we use the technique described in Section 5.3. For each verb in our list we extract all the subsequent NP and PP chunks and their heads from the chunker output. We then perform subcategorization frame learning with all subsets of these extracted potential arguments. The counts are appropriately assigned to these subsets to provide a well-defined model. Using these counts and the methods in Section 5.3 we categorize a verb as either transitive or intransitive. For simplicity, any number of arguments above zero is considered to be a candidate for transitivity.

5.4.7 Constructing the Classifi er

After we obtain all the probabilistic distributions of the features of our testing verbs, we then use C5.0 [Qui92] to construct the classifier. The data was annotated with the right classification for each verb and the classifier was run on 10% of the data using 10-fold cross validation.

5.5 Results

We tried all possible feature combinations (individual features and all possible conjunctions of those features) to explore the contributions of each feature to the reduction of the error rate. The following are the results of the best performing feature combinations.

With our base features, ACT, PASS, VBD, VBN, TRAN, and INTRAN we get the average error rate of 49.4% for 10 fold cross validation. We can see that when we add the CAUS feature, the average error decreases to 41.1%. The CAUS feature helps in 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 in addition to the

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 5.1: Results of the verb classification. Bold face results are for the best performing set of features in the classifier.

base features. The ANIM feature is an important feature that we can use to construct the classifier. When we add the 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 could be 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 as the lower performance when combining the CAUS and PART OF SPEECH features.

These experiments show that the two features proposed in this chapter: TRAN/INTRAN discovered using the likelihood ratio test for finding SF frames and the PART OF SPEECH feature provided a significant boost to the performance of the classifier.

The accuracy of the baseline classifier (picking the right argument structure at chance) was 34.5%. [MS01] calculate the expert-based upper bound at this task to be an accuracy of 86.5%.

Our best performing classifier achieves a 33.4% error rate. In comparison, [MS01] obtain an error rate of 30.2% using a much larger data set of 65M words of WSJ text. They also had less noisy data as compared to our use of noun and verb-phrase chunks (their data was parsed). We are often mislead by the putative SF of the verb and the head noun of the complement.

5.6 Conclusion

In this chapter, we discussed a technique which automatically identified the correct argument structure of a set of verbs. Our results in this paper serve as a replication and extension of the results in [MS01]. Our main contribution in this paper is to show that with reasonable accuracy, this task can be accomplished using only tagged and chunked data. In addition, we incorporate some additional features such as part-of-speech tags and the use of subcategorization frame learning as part of our classification algorithm. We exploited the distributions of selected features from the local context of the verb which was extracted from a 23M word WSJ corpus. We used C5.0 to construct a decision tree classifier using the values of those features. We were able to construct a classifier that has an error rate of 33.4%.

Chapter 6

Parsing Algorithms for Tree Adjoining Grammars

6.1 A Head-Corner Parsing Algorithm for Probabilistic TAGs

In this chapter we define a parsing algorithm for probabilistic TAGs that we use within the statistical parser that was used in this disseration (particularly in Chapter 3). The parser is a chart-based head-corner parser for general TAGs.

6.1.1 History of Head-Corner Parsing

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 [Kay89] for CF parsing (see [Sik97] for a more detailed survey). [LS91] provided the first head-driven algorithm for LTAGs which was a chart-based algorithm but it lacked any top-down prediction. [vN94] describes a Prolog implementation of a head-corner parser for LTAGs which includes top-down prediction. Significantly, [vN94] uses a different closure relation from [LS91]. 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 [vN94] algorithm. We use the same head-corner closure relation as proposed there. Our parser differs from the algorithm in [vN94] 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 [vN94] 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.

6.1.2 Head-Corner Traversal

We take the concept of head-corner traversal for TAGs as defined in [vN94]. 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 6.1, 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 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

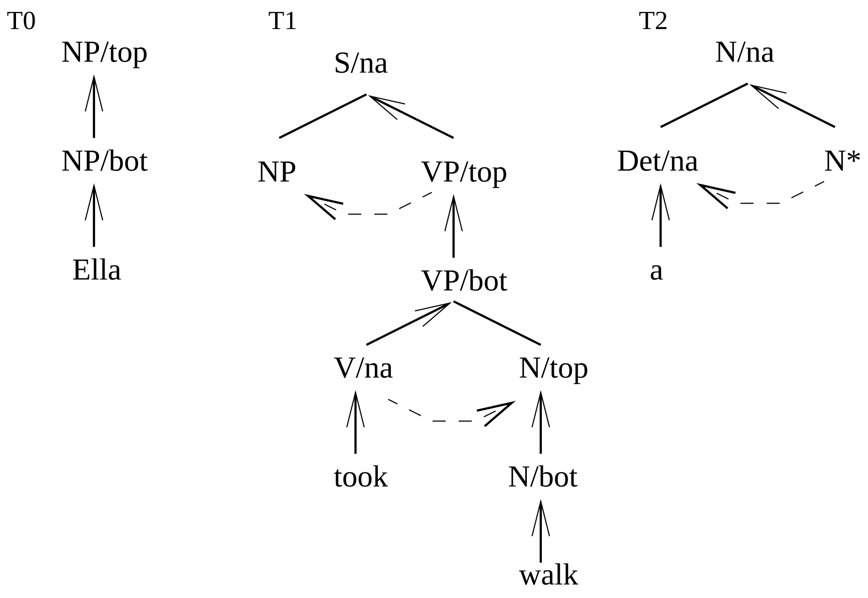


Figure 6.1: An example of head-corner traversal for parsing the sentence *Ella took a walk*.

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*.

6.1.3 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>
Heap = { s | s is State }
Agenda = { p | p is Proc }
Chart = A{len, len} with a_{i,j} = Heap
```

6.1.4 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 Heap 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 $\rightarrow$ Pr) then State $\rightarrow$ Pr = Pr

```
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__
```

Figure 6.2: Pseudo-code for the parsing algorithm

6.1.5 The Parsing Algorithm

The pseudo-code for the parsing algorithm is given in Figure 6.1.5. 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.

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. The pseudo-code for the function corresponding to *initialize headcorner* is given in Figure 6.1.5.

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.

The pseudo-code for the function corresponding to *move up* is given in Figure 6.1.5.

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.

The pseudo-code for the function corresponding to *completer* is given in Figure 6.1.5.

```
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)
}
```

Figure 6.3: Pseudo-code for the `init_head` function

```
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)
}
```

Figure 6.4: Pseudo-code for the move_up function

```
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)
}
```

Figure 6.5: Pseudo-code for the `completer` function

6.2 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 [JLT75])

The grammar factors are as follows: the parser takes $O(|A||I \cup A|Nn^6)$ worst case time and $O(|A \cup I|Nn^4)$ 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.

6.2.1 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 [MSM93]. The extraction tool [Xia99] 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 6.6 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 6.7.

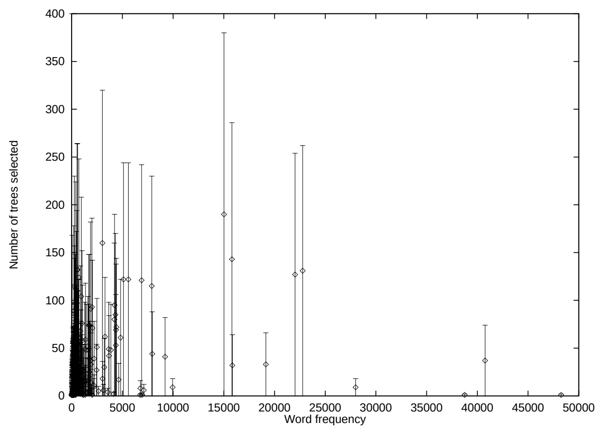


Figure 6.6: Number of trees selected plotted against words with a particular frequency. (x-axis: words of frequency *x*; y-axis: number of trees selected, error bars indicate least and most ambiguous word of a particular frequency *x*)

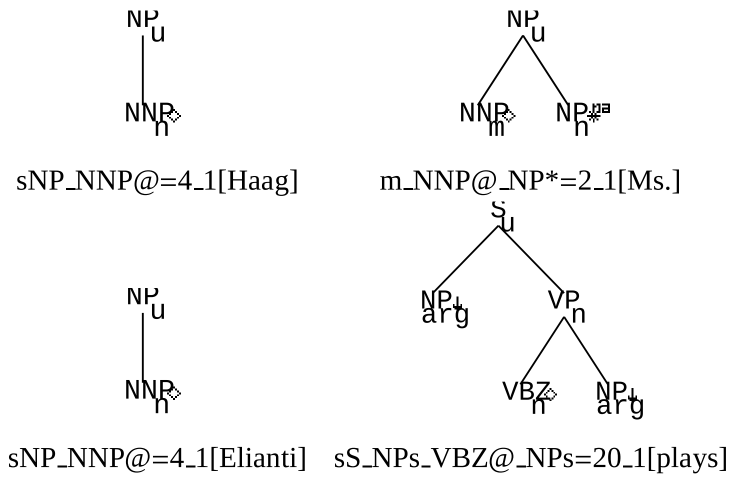


Figure 6.7: Example lexicalized elementary trees from the Treebank Grammar. They are shown in the usual notation: $\diamond$ = *anchor*, $\downarrow$= *substitution node*, $*$ = *footnode*, **na** = *null-adjunction constraint*. These trees can be combined using substitution and adjunction to parse the sentence *Ms. Haag plays Elianti*.

6.2.2 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 [JS91]:

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 6.2.1. 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 6.8 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 6.9 we plotted the number of trees selected by a sentence against 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 6.8 and a value of 0.82 for Figure 6.9. 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.

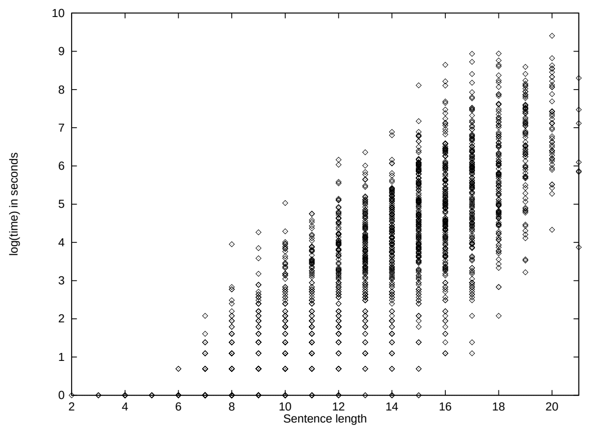


Figure 6.8: Parse times plotted against sentence length. Coefficient of determination: $R^2 = 0.65$. (x-axis: Sentence length; y-axis: log(time in seconds))

[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.

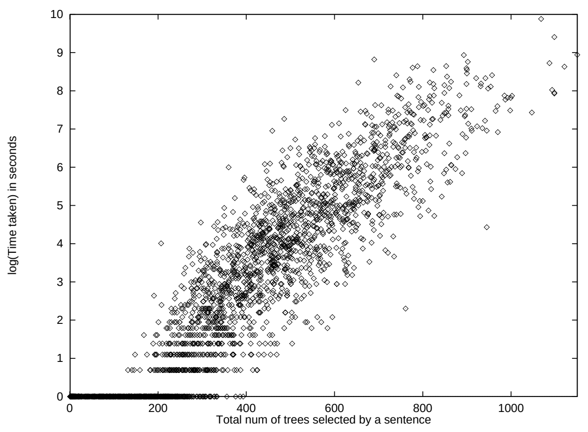


Figure 6.9: The impact of syntactic lexical ambiguity on parsing times. Log of the time taken to parse a sentence plotted against the total number of trees selected by the sentence. Coefficient of determination: $R^2 = 0.82$. (x-axis: Total number of trees selected by a sentence; y-axis: log(time) in seconds).

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 6.10. 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. This result might be surprising, in that a simple elimination of trees should not have any effect on the amount of attachment ambiguity. However, by eliminating certain elementary trees from participating in a parse, attachment ambiguities are also sometimes eliminated. Take for example the tree for a preposition like *of*. This preposition usually takes two different elementary trees: one which modifies (adjoins into) an *NP* and another that modifies a *VP*. An oracle which picks a unique tree for the preposition, in effect, eliminates one of these trees from consideration, thereby limiting the attachment ambiguity in the parser.

Figure 6.10 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 [Sri97d] 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 6.11. 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 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.

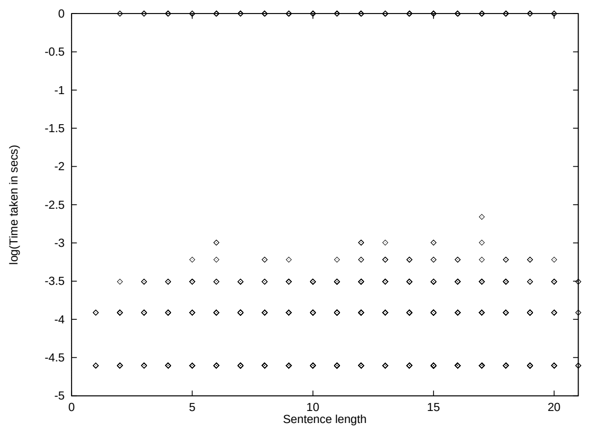


Figure 6.10: Parse times when the parser gets the correct tree for each word in the sentence (eliminating any syntactic lexical ambiguity). The parsing times for all the 2250 sentences for all lengths never goes above 1 second. (x-axis: Sentence length; y-axis: log(time) in seconds)

[2][CBVS99] 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*.

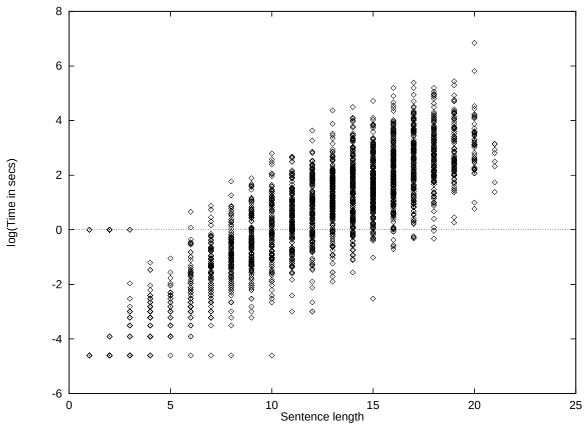


Figure 6.11: Time taken by the parser after *n*-best SuperTagging ($n = 60$). (x-axis: Sentence length; y-axis: log(time) in seconds)

6.3 Statistical Parsing of Korean

In this section we describe our experience of taking our English statistical parser and training it to work on the Korean language. We built an LTAG-based parsing system for Korean which combines corpus-based morphological analysis and tagging with a statistical parser.

The LTAG grammar we use in the parser is extracted using Lextract from the Penn Korean Treebank. The Treebank has 54,366 words, 5078 sentences. The annotation consists of a phrase structure analysis for each sentence, with head/phrase level tags as well as function tags. Each word is morphologically analyzed, where the lemma and the inflections are identified. The lemma is tagged with a part-of-speech tag (e.g., NNC, VV), and the inflections are tagged with inflectional tags (e.g., PCA, EPF, EFN).

Treebank-trained Morphological Tagger The use of lexical information plays a prominent role in statistical parsing models for English. In this paper, we extend a statistical parser that relies on bigrams of lexical dependencies to a morphologically complex language like Korean. While these types of parsers have to deal with sparse data problems, this problem is exacerbated in the case of Korean due to the fact that several base-forms of words can appear with a wide array of morphological affixes.

This problem is addressed by incorporating a statistical morphological analyzer and tagger written by Chung-hye Han, which significantly improves performance. The trigram-based morphological analyzer was trained on 91% of the treebank and tested on 9% of the treebank. This approach yielded 95.39% recall and 95.78% precision on the test data. The input to the tagger and the final output are shown below. For readability, the example has been romanized. The morphological tagger assigns part-of-speech tags but also splits the inflected form of the word into its constituent stem and affixes.

```
Input:
   {Ce-Ka} {Kwan-Cheuk} {Sa-Hang-eul} {Po-Ko-Ha-yeoss-Seup-Ni-Ta} .


Output:
   {Ce}/NPN+{Ka}/PCA {Kwan-Cheuk}/NNC {Sa-Hang}/NNC+{eul}/PCA
                    {Po-Ko-Ha}/VV+{eoss}/EPF+{Seup-Ni-Ta}/EFN ./SFN
```

Apart from the use of a specialized morphological analyzer for Korean, our methods are language independent and have been tested in previous work on the WSJ Penn English Treebank. We use Lextract to convert the Treebank (the same method is used for both the English and the Korean treebanks) into a parser derivation tree for each sentence. The statistical parsing model is then trained on these derivation trees which provide a natural domain to describe the dependencies between pairs of words in a sentence.

The performance achieved is quite comparable to state-of-the-art English statistical parsers trained on similar amounts of data. Our parser is trained on 91% of the TreeBank and tested on 9%. An off-the-shelf parser was tested on the same test set. For the sake of fair comparison, the off-the-shelf parser was converted to look as close as possible to our output. Even so, the number of node labels did not match, due to the difference in tokenization schemes for certain lexical elements such as copulas and auxiliary verbs. We thus report precision/recall for the off-the-shelf parser; we report word-to-word dependency accuracy compared with the gold standard for our parser.

	On training data	On test data
Penn	97.58	75.7
Off-the-Shelf	27.15/26.96 P/R	52.29/51.95 P/R

Table 6.1: Korean parser evaluation results

6.4 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. We also showed that the statistical parser we wrote for English can be re-trained to parse Korean. The main difference was the inclusion of a statistical morphological analyzer and tagger. The Korean parser obtained an accuracy of 75.7% when tested on the test set (of 457 sentences). This performance is better than an existing off-the-shelf Korean parser run on the same data.

Chapter 7

Conclusion

In this dissertation we described methods that use some existing linguistic resource that has been annotated by humans and add some further significant linguistic annotation by applying statistical machine learning algorithms. This chapter summarizes the results obtained in this dissertation and indicates future directions of research in the combination of labeled and unlabeled data in statistical parsing.

7.1 Summary of the Results

In Chapter 3 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 of 9695 sentences (234467 word tokens) of bracketed data, a large pool of unlabeled text and a tag dictionary of lexicalized structures for each word in this training set (based on the LTAG formalism). The algorithm presented iteratively labels the unlabeled data set with parse trees. We then trained a statistical parser on the combined set of labeled and unlabeled data. We obtained 80.02% and 79.64% labeled bracketing precision and recall respectively. The baseline model which was only trained on the 9695 sentences of labeled data performed at 72.23% and 69.12% precision and recall. These results show that training a statistical parser using our Co-training method to combine labeled and unlabeled data strongly outperforms training only on the labeled data. We also reported on an experiment which performed Co-training on two statistical parsers which had different probability models. This experiment used the entire 1M word Penn Treebank as the labeled data and a 23M word WSJ corpus as the unlabeled set. The results were not as compelling as the first experiment and the chapter discusses the various reasons for this.

In Chapter 4 we 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%.

In Chapter 5, we automatically identified the correct argument structure of a set of verbs. We exploited the distributions of some selected features from the local context of the verb which was extracted from a 23M word WSJ corpus based on part-of-speech tags and phrasal chunks alone. This annotation was minimal as compared to previous work on this task which used full parse trees. We used C5.0 to construct a decision tree classifier using the values of those features. We were able to construct a classifier that has an error rate of 33.4%. Our result compares very favorably with previous work despite using considerably less data and requiring only minimal annotation of the data.

7.2 Future Directions

In the application of co-training to parsing and further development of similar algorithms which exploit unlabeled data to improve performance in future work it would be useful to look at the following points.

- The relationship between co-training and EM bears further investigation. Tying the parameters of EM-based algorithms to reflect the available labeled data even in the face of future iterations of the EM algorithm might be one way to combine the insights of more discriminative methods such as co-training. Other methods include using discriminative objective functions as discussed in Chapter 3.

- In our experiments, unlike [BM98] we do not balance the label priors when picking new labeled examples for addition to the training data. One way to incorporate this into our algorithm would be to incorporate some form of sample selection (or active learning) into the selection of examples that are considered as labeled with high confidence [Hwa00].

- In the context of the Johns Hopkins Summer Workshop in 2002, we plan to explore the co-training of statistical parsers each of which are trained on the full Penn Treebank with the co-training algorithm exploiting a large (23M-60M word) WSJ corpus. We plan to use this combination of labeled and unlabeled data along with ideas from parser combination to produce an automatically parsed but high quality Treebank (similar in spirit to BLLIP).

There are various parts of contemporary statistical parsers that are heuristically determined without much justification other than some linguistic intuition (for picking head rules, for example) and considerations of computational limitations (pruning tricks such as those sensitive to punctuations, for example). Incorporating the learning of head-rules in a parser along with the learning of argument-adjunct decisions (using techniques given in Chapter 4) and discovering underlying argument-structure of predicates (as described in Chapter 5 based on improvements in the accuracy of the statistical parser is promising direction to move in.

Appendix A

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

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

From the literature on probabilistic context-free grammars (CFGs) we know precisely the conditions which ensure that (A.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. [Jos88] and [JS92] 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 [RJ95] 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 [Sch92]. 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 [CW97].

A.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 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 $label(A) = S$.

[1]Note that for CFGs it has been shown in [CPG83, SB97] 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.

$\mathcal{I}$ 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.

A.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 [VS87], p. 104). An example of such a function ψ is a simple Poisson distribution (A.2), which in fact was also used as the counterexample in [BT73] for CFGs, since CFGs also have the constant growth property.

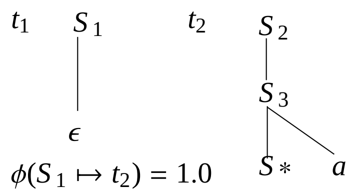
$$\psi(a^n b^n c^n d^n) = \frac{1}{e \cdot n!} \tag{A.2}$$

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 (A.3)[2].



$$\phi(S_1 \mapsto t_2) = 1.0$$
$$\begin{aligned}\phi(S_2 \mapsto t_2) &= 0.99\\ \phi(S_2 \mapsto nil) &= 0.01\\ \phi(S_3 \mapsto t_2) &= 0.98\\ \phi(S_3 \mapsto nil) &= 0.02\end{aligned} \tag{A.3}$$

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.

A.3 Conditions for Consistency

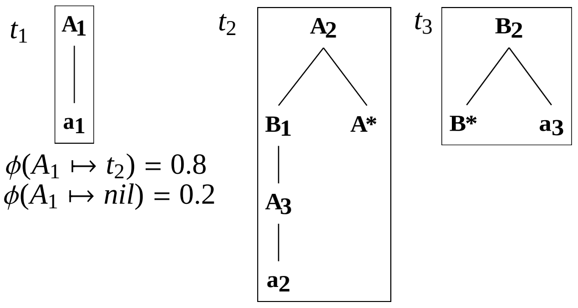
A probabilistic TAG G is *consistent* if and only if:

$$\sum_{v \in L(G)} \Pr(v) = 1 \tag{A.4}$$

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.

[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.

To explain the conditions under which a probabilistic TAG is consistent we will use the TAG in (A.5) as an example.



$\phi(A_1 \mapsto t_2) = 0.8$
$\phi(A_1 \mapsto nil) = 0.2$

$\phi(A_2 \mapsto t_2) = 0.2$
$\phi(A_2 \mapsto nil) = 0.8$
$\phi(B_1 \mapsto t_3) = 0.2$
$\phi(B_1 \mapsto nil) = 0.8$
$\phi(A_3 \mapsto t_2) = 0.4$
$\phi(A_3 \mapsto nil) = 0.6$

$\phi(B_2 \mapsto t_3) = 0.1$
$\phi(B_2 \mapsto nil) = 0.9$

(A.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 **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].

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

We then write a matrix **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.

[3]Note that **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 §A.4.

$$\mathbf{N} = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \end{array} \begin{array}{c} \begin{array}{ccccc} A_1 & A_2 & B_1 & A_3 & B_2 \end{array} \\ \left[\begin{array}{ccccc} 1.0 & 0 & 0 & 0 & 0 \\ 0 & 1.0 & 1.0 & 1.0 & 0 \\ 0 & 0 & 0 & 0 & 1.0 \end{array}\right] \end{array}$$

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 [BT73] 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 [Har63].

Theorem A.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. [BT73, Sou74]*

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 [HJ85, Wet80]).

[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 [Har63] for details on how this special case can be solved.

Theorem A.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 [HJ85, Wet80])*

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 (A.5) we computed the following stochastic expectation matrix:

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

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 (A.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.

[5]We compute $\mathcal{M}^{2^2}$ and subsequently only successive powers of 2 because Theorem A.3.2 holds for any $n' > n$. This permits us to use a single matrix at each step in the algorithm.

Now consider the grammar (A.3) we had considered in Section A.2. The value of $\mathcal{M}$ for that grammar is computed to be:

$$\mathcal{M}_{(A.3)} = \begin{array}{c|ccc} & S_1 & S_2 & S_3 \\ \hline S_1 & 0 & 1.0 & 1.0 \\ S_2 & 0 & 0.99 & 0.99 \\ S_3 & 0 & 0.98 & 0.98 \end{array}$$

The eigenvalues for the expectation matrix $\mathcal{M}$ computed for the grammar (A.3) are 0, 1.97 and 0. The largest eigenvalue is greater than 1 and this confirms (A.3) to be an inconsistent grammar.

A.4 TAG Derivations and Branching Processes

To show that Theorem A.3.1 in Section A.3 holds for any probabilistic TAG, it is sufficient to show that the derivation process in TAGs is a Galton-Watson branching process.

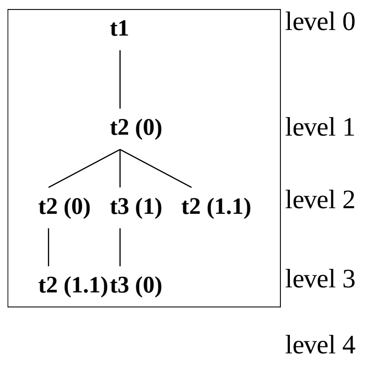
A Galton-Watson branching process [Har63] is simply a model of processes that have objects that can produce additional objects of the same kind, i.e. recursive processes, with certain properties. There is an initial set of objects in the 0-th generation which produces with some probability a first generation 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, $\ldots$ 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.

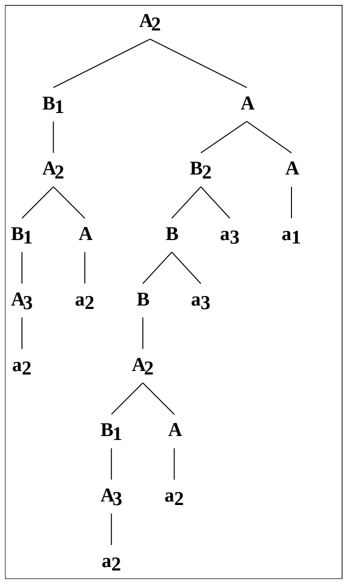
2. The number of objects born to a parent object does not depend on how many other objects are present at the same level.

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.



(A.6)



(A.7)

The assumptions made about the generating process intuitively holds for probabilistic TAGs. (A.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 (A.5)[6]. The parse tree derived from such a sequence is shown in Fig. A.7. In the derivation tree (A.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 (A.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 A.3.1 in

[6]The numbers in parentheses next to the tree names are node addresses where each tree has adjoined into its parent. Recall the defi nition of node addresses in Section A.1.

Section A.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 $s_1, \ldots, s_k$ 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 (A.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.

$$\begin{aligned}
& g_1(s_1, \ldots, s_5) = \\
& \quad \phi(A_1 \mapsto t_2) \cdot s_2 \cdot s_3 \cdot s_4 + \phi(A_1 \mapsto nil) \\
& g_2(s_1, \ldots, s_5) = \\
& \quad \phi(A_2 \mapsto t_2) \cdot s_2 \cdot s_3 \cdot s_4 + \phi(A_2 \mapsto nil) \\
& g_3(s_1, \ldots, s_5) = \\
& \quad \phi(B_1 \mapsto t_3) \cdot s_5 + \phi(B_1 \mapsto nil) \\
& g_4(s_1, \ldots, s_5) = \\
& \quad \phi(A_3 \mapsto t_2) \cdot s_2 \cdot s_3 \cdot s_4 + \phi(A_3 \mapsto nil) \\
& g_5(s_1, \ldots, s_5) = \\
& \quad \phi(B_2 \mapsto t_3) \cdot s_5 + \phi(B_2 \mapsto nil)
\end{aligned}$$

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

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

For the grammar in (A.5) we get the following level generating functions.

$$
\begin{aligned}
&G_0(s_1, \ldots, s_5) = s_1 \\
&G_1(s_1, \ldots, s_5) = g_1(s_1, \ldots, s_5) \\
&\quad = \quad \phi(A_1 \mapsto t_2) \cdot s_2 \cdot s_3 \cdot s_4 + \phi(A_1 \mapsto nil) \\
&\quad = \quad 0.8 \cdot s_2 \cdot s_3 \cdot s_4 + 0.2 \\
&G_2(s_1, \ldots, s_5) = \\
&\qquad \phi(A_2 \mapsto t_2)[g_2(s_1, \ldots, s_5)][g_3(s_1, \ldots, s_5)] \\
&\qquad\quad [g_4(s_1, \ldots, s_5)] + \phi(A_2 \mapsto nil) \\
&\quad = \quad 0.08 s_2^2 s_3^2 s_4^2 s_5 + 0.03 s_2^2 s_3^2 s_4^2 + 0.04 s_2 s_3 s_4 s_5 + \\
&\qquad\quad 0.18 s_2 s_3 s_4 + 0.04 s_5 + 0.196 \\
&\ldots
\end{aligned}
$$

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. [BT73]).

$$m_{i,j} = \left. \frac{\partial g_i(s_1, \ldots, s_k)}{\partial s_j} \right|_{s_1, \ldots, s_k = 1} \tag{A.8}$$

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 A.3.1 used in Section A.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 [CPG83] and [SB97] 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.

A.5 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 2 probability computation from probabilistic TAGs.

Appendix B

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 $O(n^6)$ 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.

B.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 [WW89, JL91, Sto95]. 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 [RJ95], which is relevant because recent work on *structured* language models [CEJ+97] 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 [Sri96].

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 [Sch92] in probabilistic TAGs to be used for language modeling.

B.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 $label(N)$ to be the label of the node, which is either a terminal from Σ or the empty string ϵ. For each other node N, *label*(N) is an element from $\mathcal{N}$.

At a node N in a tree such that *label*$(N) \in \mathcal{N}$ 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 *label*$(N) \in \mathcal{N}$,

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

For each non-leaf node N we construct the string $cdn(N) = \widehat{N_1} \cdots \widehat{N_m}$ from the (ordered) list of children nodes $N_1, \ldots, N_m$ by defining, for each d such that $1 \leq d \leq m$, $\widehat{N_d} = label(N_d)$ in case $label(N_d) \in \Sigma \cup \{\epsilon\}$, 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 $cdn(F_t) = \perp$ for each auxiliary tree t. We set

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

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 $dft(N)$ 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 $dft(N_1 \ldots N_m)$ if and only if there is a d $(1 \leq d \leq m)$ such that $dft(N_d)$.

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

B.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. B.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. B.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.

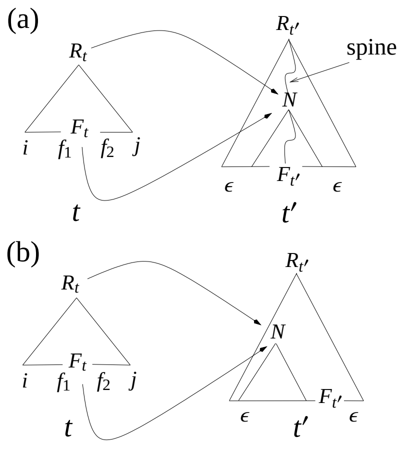


Figure B.1: Wrapping in auxiliary trees with empty yield

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. B.1 now generalize to those in Fig. B.2.

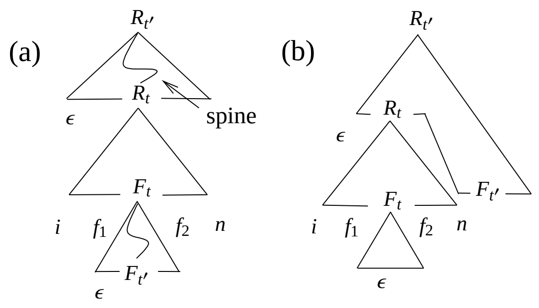


Figure B.2: Wrapping of auxiliary trees when computing the prefix probability

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 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 [Lan88]. 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.

B.4 Computing Prefi x Probabilities

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

B.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:

$$P([t, i, j, f_1, f_2]) = P([R_t, i, j, f_1, f_2]); \tag{B.1}$$

$$P([\alpha N, i, j, -, -]) = \tag{B.2}$$
$$\sum_{k (i \leq k \leq j)} P([\alpha, i, k, -, -]) \cdot P([N, k, j, -, -]),$$
$$\text{if } \alpha \neq \epsilon \wedge \neg dft(\alpha N);$$

$$P([\alpha N, i, j, f_1, f_2]) = \tag{B.3}$$
$$\sum_{k (i \leq k \leq f_1)} P([\alpha, i, k, -, -]) \cdot P([N, k, j, f_1, f_2]),$$
$$\text{if } \alpha \neq \epsilon \wedge dft(N);$$

$$P([\alpha N, i, j, f_1, f_2]) = \tag{B.4}$$
$$\sum_{k (f_2 \leq k \leq j)} P([\alpha, i, k, f_1, f_2]) \cdot P([N, k, j, -, -]),$$
$$\text{if } \alpha \neq \epsilon \wedge dft(\alpha);$$

$$P([N, i, j, f_1, f_2]) = \tag{B.5}$$
$$\phi(\mathbf{nil}, N) \cdot P([cdn(N), i, j, f_1, f_2]) +$$

$$\sum_{f_1', f_2' (i \leq f_1' \leq f_1 \wedge f_2 \leq f_2' \leq j)} P([cdn(N), f_1', f_2', f_1, f_2]) \cdot \sum_{t \in \mathcal{A}} \phi(t, N) \cdot P([t, i, j, f_1', f_2']),$$
$$\text{if } N \in V \wedge dft(N);$$

$$P([N, i, j, -, -]) = \tag{B.6}$$
$$\phi(\mathbf{nil}, N) \cdot P([cdn(N), i, j, -, -]) + \sum_{f_1', f_2' (i \leq f_1' \leq f_2' \leq j)} P([cdn(N), f_1', f_2', -, -]) \cdot \sum_{t \in \mathcal{A}} \phi(t, N) \cdot P([t, i, j, f_1', f_2']),$$
$$\text{if } N \in V \wedge \neg dft(N);$$

$$P([a, i, j, -, -]) = \tag{B.7}$$
$$\delta(i + 1 = j \wedge a_j = a) + \delta(i = j = n);$$

$$P([\perp, i, j, f_1, f_2]) = \delta(i = f_1 \wedge j = f_2); \tag{B.8}$$

$$P([\epsilon, i, j, -, -]) = \delta(i = j). \tag{B.9}$$

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 (B.1) through (B.6). In equations (B.5) and (B.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 (B.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.

B.4.2 Terminating equations

In general, the recursive equations (B.1) to (B.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 nontermination. 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:

$(i', j', f_1', f_2') \approx (i, j, f_1, f_2)$ if and only if

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

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:

$$
\begin{aligned}
P([t, i, j, f_1, f_2]) \ = \\
\sum_{t' \in \mathcal{A}, f'_1, f'_2 ((i, j, f'_1, f'_2) \approx (i, j, f_1, f_2))} P_{low}([t, i, j, f_1, f_2],\ [t', f'_1, f'_2]).
\end{aligned} \tag{B.10}
$$

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

$$
\begin{aligned}
P([t, i, j, -, -]) \ = \\
\sum_{t' \in \{t\} \cup \mathcal{A}, f \in \{-, i, j\}} P_{low}([t, i, j, -, -],\ [t', f, f]).
\end{aligned} \tag{B.11}
$$

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 (B.29) and (B.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.

$$
\begin{aligned}
&P_{low}([t,\ i,\ j,\ f_1,\ f_2],\ [t', f_1', f_2'])\ = \\
&\quad P_{low}([R_t,\ i,\ j,\ f_1,\ f_2],\ [t', f_1', f_2']) + \\
&\quad \delta((t, f_1, f_2) = (t', f_1', f_2')) \cdot \\
&\qquad\quad P_{split}([R_t,\ i,\ j,\ f_1,\ f_2]);
\end{aligned} \tag{12}
$$

$$
\begin{aligned}
&P_{low}([\alpha N, i, j, -, -],\ [t, f_1', f_2'])\ = \\
&\quad P_{low}([\alpha, i, j, -, -],\ [t, f_1', f_2']) \cdot \\
&\qquad\quad P([N, j, j, -, -]) + \\
&\quad P([\alpha, i, i, -, -]) \cdot \\
&\qquad\quad P_{low}([N, i, j, -, -],\ [t, f_1', f_2']), \\
&\quad \text{if } \alpha \neq \epsilon \wedge \neg dft(\alpha N);
\end{aligned} \tag{13}
$$

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

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

$$
P_{low}([N, i, j, f_1, f_2],\ [t, f_1', f_2'])\ = \tag{16}
$$

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

$$P_{low}([N, i, j, -, -],\ [t, f_1', f_2']) = \tag{17}$$
$$\phi(\mathbf{nil}, N) \cdot$$
$$\quad P_{low}([cdn(N), i, j, -, -],\ [t, f_1', f_2']) +$$
$$P_{low}([cdn(N), i, j, -, -],\ [t, f_1', f_2']) \cdot$$
$$\quad \textstyle\sum_{t' \in \mathcal{A}} \phi(t', N) \cdot P([t', i, j, i, j]) +$$
$$\sum_{f_1'', f_2'' (f_1'' = f_2'' = i \vee f_1'' = f_2'' = j)} P([cdn(N), f_1'', f_2'', -, -]) \cdot$$
$$\sum_{t' \in \mathcal{A}} \phi(t', N) \cdot P_{low}([t', i, j, f_1'', f_2''],\ [t, f_1', f_2']),$$
$$\text{if } N \in V \wedge \neg dft(N);$$

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

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

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

The definition of P_{low} parallels the one of P given in §B.4.1. In (B.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 (B.18), (B.19) and (B.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 §B.4.1.

$$P_{split}([\alpha N, i, j, -, -]) = \tag{B.21}$$
$$\sum_{k(i<k<j)} P([\alpha, i, k, -, -]) \cdot P([N, k, j, -, -]) +$$
$$P_{split}([\alpha, i, j, -, -]) \cdot P([N, j, j, -, -]) +$$
$$P([\alpha, i, i, -, -]) \cdot P_{split}([N, i, j, -, -]),$$
$$\text{if } \alpha \neq \epsilon \wedge \neg dft(\alpha N);$$

$$P_{split}([\alpha N, i, j, f_1, f_2]) = \tag{B.22}$$
$$\sum_{k(i<k\leq f_1 \wedge k<j)} P([\alpha, i, k, -, -]) \cdot P([N, k, j, f_1, f_2]) +$$
$$\delta(f_1 = j) \cdot P_{split}([\alpha, i, j, -, -]) \cdot$$
$$P([N, j, j, f_1, f_2]) +$$
$$P([\alpha, i, i, -, -]) \cdot P_{split}([N, i, j, f_1, f_2]),$$
$$\text{if } \alpha \neq \epsilon \wedge dft(N);$$

$$P_{split}([\alpha N, i, j, f_1, f_2]) = \tag{B.23}$$
$$\sum_{k(i<k \wedge f_2\leq k<j)} P([\alpha, i, k, f_1, f_2]) \cdot P([N, k, j, -, -]) +$$
$$P_{split}([\alpha, i, j, f_1, f_2]) \cdot P([N, j, j, -, -]) +$$
$$\delta(i = f_2) \cdot P([\alpha, i, i, f_1, f_2]) \cdot$$
$$P_{split}([N, i, j, -, -]),$$
$$\text{if } \alpha \neq \epsilon \wedge dft(\alpha);$$

$$P_{split}([N, i, j, f_1, f_2]) = \tag{B.24}$$
$$\phi(\mathbf{nil}, N) \cdot P_{split}([cdn(N), i, j, f_1, f_2]) +$$
$$\sum_{\substack{f_1', f_2' \ (i \leq f_1' \leq f_1 \wedge f_2 \leq f_2' \leq j \wedge \\ (f_1', f_2') \neq (i, j) \wedge (f_1', f_2') \neq (f_1, f_2))}} P([cdn(N), f_1', f_2', f_1, f_2]) \cdot$$
$$\sum_{t \in \mathcal{A}} \phi(t, N) \cdot P([t, i, j, f_1', f_2']) +$$
$$P_{split}([cdn(N), i, j, f_1, f_2]) \cdot$$
$$\sum_{t \in \mathcal{A}} \phi(t, N) \cdot P([t, i, j, i, j]),$$

$$\text{if } N \in V \wedge dft(N);$$

$$P_{split}([N, i, j, -, -]) \ = \tag{B.25}$$

$$\phi(\mathbf{nil}, N) \cdot P_{split}([cdn(N), i, j, -, -]) \ +$$

$$\sum_{\substack{f'_1, f'_2 \ \ (i \leq f'_1 \leq f'_2 \leq j \wedge (f'_1, f'_2) \neq (i, j) \wedge \\ \neg(f'_1 = f'_2 = i \vee f'_1 = f'_2 = j))}} P([cdn(N), f'_1, f'_2, -, -]) \ \cdot$$

$$\sum_{t \in \mathcal{A}} \phi(t, N) \cdot P([t, i, j, f'_1, f'_2]) \ +$$

$$P_{split}([cdn(N), i, j, -, -]) \ \cdot$$

$$\sum_{t \in \mathcal{A}} \phi(t, N) \cdot P([t, i, j, i, j]),$$

$$\text{if } N \in V \wedge \neg dft(N);$$

$$P_{split}([a, i, j, -, -]) = \delta(i + 1 = j \wedge a_j = a); \tag{B.26}$$

$$P_{split}([\perp, i, j, f_1, f_2]) = 0; \tag{B.27}$$

$$P_{split}([\epsilon, i, j, -, -]) = 0. \tag{B.28}$$

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:

$$P_{outer}([t, i, j, f_1, f_2], \ [t', f'_1, f'_2]) \ = \tag{B.29}$$
$$\frac{P_{low}([t, i, j, f_1, f_2], \ [t', f'_1, f'_2])}{P_{split}([R_{t'}, i, j, f'_1, f'_2])};$$

$$P_{outer}([\alpha, i, j, f_1, f_2], \ [t', f'_1, f'_2]) \ = \tag{B.30}$$
$$\frac{P_{low}([\alpha, i, j, f_1, f_2], \ [t', f'_1, f'_2])}{P_{split}([R_{t'}, i, j, f'_1, f'_2])}.$$

We can now eliminate the infinite recursion that arises in (B.10) and (B.11) by rewriting $P([t, i, j, f_1, f_2])$ in terms of P_{outer}:

$$P([t, i, j, f_1, f_2]) \ = \tag{B.31}$$

$$\sum_{t' \in \mathcal{A}, \, f'_1, f'_2 ((i, j, f'_1, f'_2) \approx (i, j, f_1, f_2))} P_{outer}([t, i, j, f_1, f_2], \ [t', f'_1, f'_2]) \ \cdot$$
$$P_{split}([R_{t'}, i, j, f'_1, f'_2]);$$

$$P([t, i, j, -, -]) = \sum_{t' \in \{t\} \cup \mathcal{A},\, f \in \{-, i, j\}} P_{outer}([t, i, j, -, -],\ [t', f, f]) \cdot P_{split}([R_{t'}, i, j, f, f]). \tag{B.32}$$

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

In summary, terminating computation of prefix probabilities should be based on equations (B.31) and (B.32), which replace (B.1), along with equations (B.2) to (B.9) and all the equations for P_{split}.

B.4.3 Off-line Equations

In this section we derive equations for function P_{outer} introduced in §B.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 (B.1) to (B.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 $f_1, f_2 \notin \{i, j, -\}$ (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 (B.12) to (B.20) according to (B.29) and (B.30). For instance, combining (B.12) and (B.29) gives (using the above assumptions on f_1 and f_2):

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

Also, if $\alpha \neq \epsilon$ and $dft(N)$, combining (B.14) and (B.30) gives (again, using previous assumptions on f_1 and f_2; note that the H_α's are known terms here):

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

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.

B.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 $O(n^6)$, where n is the length of the input prefix.

All the quantities introduced in §B.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 [Sto95]. 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, §B.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 (B.31) and (B.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 $O(n^5)$.

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.

B.6 Computing the Off-line Probabilities

To compute the off-line probabilities we simply use the results already shown in Chapter A. 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.

B.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 2.5.2.

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 $O(n^6)$ 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.

Bibliography

[Abn97] Steve Abney. Part of speech tagging and partial parsing. In S. Young and G. Bloothooft, editors, *Corpus based methods in language and speech*, pages 118–136. Dordrecht: Kluwer, 1997.

[ASS99] S. Abney, R. Schapire, and Y. Singer. Boosting applied to tagging and pp attachment, 1999.

[BAF+91] E. Black, S. Abney, D. Flickinger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. A procedure for quantitatively comparing the syntactic coverage of english grammars. In *Proc. DARPA Speech and Natural Language Workshop*, pages 306–311. Morgan Kaufmann, 1991.

[BC97] Ted Briscoe and John Carroll. Automatic extraction of subcategorization from corpora. In *Proceedings of the 5th ANLP Conference*, pages 356–363, Washington, D.C., 1997. ACL.

[BD77] Peter Bickel and Kjell Doksum. *Mathematical Statistics*. Holden-Day Inc., 1977.

[BJL+93] Ezra Black, Fred Jelinek, John Lafferty, David Magerman, Robert Mercer, and Salim Roukos. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics (ACL)*, 1993.

[BM98] Avrim Blum and Tom Mitchell. Combining Labeled and Unlabeled Data with Co-Training. In *In Proc. of 11th Annual Conf. on Comp. Learning Theory (COLT)*, pages 92–100, 1998.

[Bod01] Rens Bod. What is the minimal set of fragments that achieves maximal parse accuracy. In *Proceedings of ACL-2001*, Toulouse, France, 2001.

[Bre91] Michael Brent. Automatic acquisition of subcategorization frames from untagged text. In *Proceedings of the 29th Meeting of the ACL*, pages 209–214, Berkeley, CA, 1991.

[Bre93] Michael Brent. From grammar to lexicon: unsupervised learning of lexical syntax. *Computational Linguistics*, 19(3):243–262, 1993.

[Bre94] Michael Brent. Acquisition of subcategorization frames using aggregated evidence from local syntactic cues. *Lingua*, 92:433–470, 1994. Reprinted in Acquisition of the Lexicon, L. Gleitman and B. Landau (Eds.). MIT Press, Cambridge, MA.

[Bri97] E. Brill. Unsupervised learning of disambiguation rules for part of speech tagging. In *Natural Language Processing Using Very Large Corpora*. Kluwer Academic Press, 1997.

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

[BV98] Roberto Basili and Michele Vindigni. 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, Apr 24 1998.

[CAL94] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[CB95] M. Collins and J. Brooks. Prepositional phrase attachment through a backed-off model. In D. Yarowsky and K. Church, editors, *Proc. of the 3rd Workshop on Very Large Corpora*, pages 27–38, 1995.

[CBVS99] John Chen, Srinivas Bangalore, and K. Vijay-Shanker. New models for improving supertag disambiguation. In *Proceedings of the 9th Conference of the European Chapter of the Association for Computational Linguistics*, Bergen, Norway, 1999.

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

[Chi00] David Chiang. Statistical Parsing with an Automatically-Extracted Tree Adjoining Grammar. In *Proc. of ACL-2000*, 2000.

[Chi01] David Chiang. Statistical parsing with an automatically extracted tree adjoining grammar. In *Data Oriented Parsing*. CSLI, 2001. In this volume.

[CHRT99] M. Collins, J. Hajic, L. Ramshaw, and C. Tillmann. A statistical parser for czech. In *In Proceedings of the 37th Annual Meeting of the ACL*, College Park, Maryland, 1999.

[CJ98] C. Chelba and F. Jelinek. Exploiting syntactic structure for language modeling. In *Proc of COLING-ACL '98*, pages 225–231, Montreal, 1998.

[CKPS92] D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun. A practical part-of-speech tagger. In *Proc. of 3rd ANLP Conf.*, Trento, Italy, 1992. ACL.

[CM98] John Carroll and Guido Minnen. Can subcategorisation probabilities help a statistical parser. In *Proceedings of the 6th ACL/SIGDAT Workshop on Very Large Corpora (WVLC-6)*, Montreal, Canada, 1998.

[Col97] M. Collins. Three generative, lexicalized models for statistical parsing. In *Proc. of 35th ACL and 8th EACL*, 1997.

[Col98] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1998.

[Col00] Michael Collins. Discriminative reranking for natural language parsing. In *Proc. 17th International Conf. on Machine Learning*, pages 175–182. Morgan Kaufmann, San Francisco, CA, 2000.

[CP82] K. Church and R. Patil. Coping with syntactic ambiguity or how to put the block in the box on the table. *Computational Linguistics*, 8:139–149, 1982.

[CPG83] R. Chaudhari, S. Pham, and O. N. Garcia. Solution of an open problem on probabilistic grammars. *IEEE Transactions on Computers*, C-32(8):748–750, August 1983.

[CR98a] G. Carroll and M. Rooth. Valence Induction with a Head-Lexicalized PCFG. `http://xxx.lanl.gov/abs/cmp-lg/9805001`, May 1998.

[CR98b] Glenn Carroll and Mats Rooth. Valence induction with a head-lexicalized PCFG. In *Proceedings of the 3rd Conference on Empirical Methods in Natural Language Processing (EMNLP 3)*, Granada, Spain, 1998.

[CS99] Michael Collins and Yoram Singer. Unsupervised Models for Named Entity Classification. In *In Proc. of WVLC/EMNLP-99*, pages 100–110, 1999.

[CVS00] John Chen and K. Vijay-Shanker. Automated Extraction of TAGs from the Penn Treebank. In *Proc. of the 6th International Workshop on Parsing Technologies (IWPT-2000), Italy*, 2000.

[CW97] J. Carroll and D. Weir. Encoding frequency information in lexicalized grammars. In *Proc. 5th Int'l Workshop on Parsing Technologies IWPT-97*, Cambridge, Mass., 1997.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum-likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistics Society*, 39(1):1–38, 1977. Series B.

[Dun93] Ted Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74, March 1993.

[EC96] Murat Ersan and Eugene Charniak. A statistical syntactic disambiguation program and what it learns. In S. Wermter, E. Riloff, and G. Scheler, 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, 1996.

[ED96] Sean P. Engelson and Ido Dagan. Sample Selection in Natural Language Learning. In S. Wermter, E. Riloff, and G. Scheler, editors, *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing*, pages 230–245. Springer-Verlag, 1996.

[Eis96] Jason Eisner. An empirical comparison of probability models for dependency grammar. Technical Report IRCS-96-11, Institute for Research in Cognitive Science, Univ. of Pennsylvania, 1996.

[Elw94] David Elworthy. Does baum-welch re-estimation help taggers? In *Proceedings of 4th ACL Conf on ANLP*, pages 53–58, Stuttgart, October 13-15 1994.

[ES99] J. Eisner and G. Satta. Efficient parsing for bilexical context-free grammars and head automaton grammars, 1999.

[FW96] E. W. Fong and D. Wu. Learning restricted probabilistic link grammars. In S. Wermter, E. Riloff, and G. Scheler, editors, *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing*, pages 173–187. Springer-Verlag, 1996.

[Gil01] Daniel Gildea. Corpus variation and parser performance. In *Proceedings of 2001 Conference on Empirical Methods in Natural Language Processing*, Pittsburgh, PA, 2001.

[Goo96] Joshua Goodman. Parsing algorithms and metrics. In Aravind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 177–183, San Francisco, 1996. Morgan Kaufmann Publishers.

[GZ00] Sally Goldman and Yan Zhou. Enhancing supervised learning with unlabeled data. In *Proceedings of ICML'2000*, Stanford University, June 29–July 2 2000.

[Haj98] Jan Hajič. Building a syntactically annotated corpus: The prague dependency treebank. In *Issues of Valency and Meaning*, pages 106–132. Karolinum, Praha, 1998.

[Har63] T. E. Harris. *The Theory of Branching Processes*. Springer-Verlag, Berlin, 1963.

[HB00] J. Henderson and E. Brill. Bagging and boosting a treebank parser. In *In Proceedings of NAACL 2000*, Seattle, WA, 2000.

[Hen98] John C. Henderson. Exploiting diversity for natural language processing. In *AAAI/IAAI*, page 1174, 1998.

[HH98] Jan Hajič and Barbora Hladk´a. Tagging inflective languages: Prediction of morphological categories for a rich, structured tagset. In *Proceedings of COLING-ACL 98*, Universit´e de Montr´eal, Montr´eal, pages 483–490, 1998.

[Hin00] G. Hinton. Training products of experts by minimizing contrastive divergence, 2000.

[HJ85] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, 1985.

[Hoo01] Lars Hoogweg. Extending dop with insertion. In *Data Oriented Parsing*. CSLI, 2001. unpublished ms.

[Hwa98] R. Hwa. An Empirical Evaluation of Probabilistic Lexicalized Tree Insertion Grammars. In *Proc. of COLING-ACL '98*, pages 557–563, Montreal, Canada, 1998.

[Hwa99] R. Hwa. Supervised grammar induction using training data with limited constituent information. In *Proceedings of the 37th Annual Meeting of the ACL*, pages 73–79, June 1999.

[Hwa00] Rebecca Hwa. Sample selection for statistical grammar induction. In *Proceedings of EMNLP/VLC-2000*, pages 45–52, 2000.

[Hwa01] R. Hwa. On minimizing training corpus for parser acquisition. In *Proceedings of the Fifth Computational Natural Language Learning Workshop*, July 2001.

[JL91] F. Jelinek and J. Lafferty. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–323, 1991.

[JLT75] Aravind K. Joshi, L. Levy, and M. Takahashi. Tree Adjunct Grammars. *Journal of Computer and System Sciences*, 1975.

[Jos85] Aravind K. Joshi. Tree Adjoining Grammars: How much context Sensitivity is required to provide a reasonable structural description. In D. Dowty, I. Karttunen, and A. Zwicky, editors, *Natural Language Parsing*, pages 206–250. Cambridge University Press, Cambridge, U.K., 1985.

[Jos88] A. K. Joshi. An introduction to tree adjoining grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*. John Benjamins, Amsterdam, 1988.

[Jos90] Aravind Joshi. Processing crossed and nested dependencies: An automaton perspective on the psycholinguistic results. *Language and Cognitive Processes*, 5(1):1–27, 1990.

[JS91] A. Joshi and Y. Schabes. Tree adjoining grammars and lexicalized grammars. In M. Nivat and A. Podelski, editors, *Tree automata and languages*. North-Holland, 1991.

[JS92] A. K. Joshi and Y. Schabes. Tree-adjoining grammar and lexicalized grammars. In M. Nivat and A. Podelski, editors, *Tree automata and languages*, pages 409–431. Elsevier Science, 1992.

[JS97] Aravind Joshi and Yves Schabes. Tree adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages and Automata*. Springer-Verlag, 1997.

[Kat87] Slava Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35:400–401, 1987.

[Kay89] Martin Kay. Head driven parsing. In *Proc. of IWPT '89*, pages 52–62, Pittsburgh, PA, 1989.

[KGM00] A. Korhonen, G. Gorrell, and D. McCarthy. Statistical filtering and subcategorization frame acquisition. In *Proceedings of EMNLP 2000*, 2000.

[Lan88] B. Lang. Parsing incomplete sentences. In *Proc. of the 12th International Conference on Computational Linguistics*, volume 1, pages 365–371, Budapest, 1988.

[Lap99] Maria Lapata. Acquiring lexical generalizations from corpora: A case study for diathesis alternations. In *Proceedings of 37th Meeting of ACL*, pages 397–404, 1999.

[LB99] Maria Lapata and Chris Brew. Using subcategorization to resolve verb class ambiguity. In Pascale Fung and Joe Zhou, editors, *Proceedings of WVLC/EMNLP*, pages 266–274, 21-22 June 1999.

[Lev93] Beth Levin. *English Verb Classes and Alternations*. Chicago University Press, Chicago, IL, 1993.

[LS91] Alberto Lavelli and Giorgio Satta. Bidirectional parsing of Lexicalized Tree Adjoining Grammars. In *Proc. 5th EACL*, Berlin, Germany, April 1991.

[LST92] J. Lafferty, D. Sleator, and D. Temperley. Grammatical trigrams: A probabilistic model of link grammar. In *Proc. of the AAAI Conf. on Probabilistic Approaches to Natural Language*, 1992.

[LY90] K. Lari and S. J. Young. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[Mag95] David Magerman. Statistical decision tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 276–283, 1995.

[Man93] Christopher D. Manning. Automatic acquisition of a large subcategorization dictionary from corpora. In *Proceedings of the 31st Meeting of the ACL*, pages 235–242, Columbus, Ohio, 1993.

[Mar95] C. de Marcken. Lexical heads, phrase structure and the induction of grammar. In D. Yarowsky and K. Church, editors, *Proceedings of the Third Workshop of Very Large Corpora*, pages 14–26, MIT, Cambridge, MA, 1995.

[MCF+98] S. Miller, M. Crystal, H. Fox, L. Ramshaw, R. Schwartz, R. Stone, R. Weischedel, and t Group. Algorithms that learn to extract information–bbn: Description of the sift system as used for muc, 1998.

[Mer94] B. Merialdo. Tagging english text with a probabilistic model. *Computational Linguistics*, 20(2):155–172, 1994.

[MK98] Diana McCarthy and Anna Korhonen. Detecting verbal participation in diathesis alternations. In *Proceedings of COLING/ACL-1998. Student Session*, pages 1493–1495, 1998.

[MS01] Paola Merlo and Suzanne Stevenson. Automatic verb classification based on statistical distribution of argument structure. *Computational Linguistics*, 27(3):373–408, 2001.

[MSM93] M. Marcus, B. Santorini, and M. Marcinkiewiecz. Building a large annotated corpus of english. *Computational Linguistics*, 19(2):313–330, 1993.

[NG00] Kamal Nigam and Rayid Ghani. Analyzing the effectiveness and applicability of co-training. In *Proc. of Ninth International Conference on Information and Knowledge (CIKM-2000)*, 2000.

[Nig01] Kamal Nigam. *Using Unlabeled Data to Improve Text Classification*. PhD thesis, Computer Science Department, Carnegie Mellon University, 2001. Technical Report CMU-CS-01-126.

[NMTM99] Kamal Nigam, Andrew McCallum, Sebastian Thrun, and Tom Mitchell. Text Classification from Labeled and Unlabeled Documents using EM. *Machine Learning*, 1(34), 1999.

[NSS98] Mark-Jan Nederhof, Anoop Sarkar, and Giorgio Satta. Prefix probabilities from probabilistic tree adjoining grammars. In *Proceedings of COLING-ACL 1998*, Montreal, 1998.

[PPG$^+$94] S. Della Pietra, V. Della Pietra, J. Gillett, J. Lafferty, H. Printz, and L. Ureš. Inference and estimation of a long-range trigram model. In R. Carrasco and J. Oncina, editors, *Proc. of ICGI-94*. Springer-Verlag, 1994.

[PS92] Fernando Pereira and Yves Schabes. Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 1992.

[Qui92] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Series in Machine Learning. Morgan Kaufmann, San Mateo, CA, 1992.

[Rat96] A. Ratnaparkhi. A Maximum Entropy Part-Of-Speech Tagger. In *Proc. of the Empirical Methods in Natural Language Processing Conference*, University of Pennsylvania, 1996.

[Rat98] Adwait Ratnaparkhi. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, University of Pennsylvania, 1998.

[Res92] P. Resnik. Probabilistic tree-adjoining grammars as a framework for statistical natural language processing. In *Proc. of COLING '92*, volume 2, pages 418–424, Nantes, France, 1992.

[RJ95] O. Rambow and A. Joshi. A formal look at dependency grammars and phrase-structure grammars, with special consideration of word-order phenomena. In Leo Wanner, editor, *Current Issues in Meaning-Text Theory*. Pinter, London, 1995.

[RRR94] Adwait Ratnaparkhi, Jeff Reynar, and Salim Roukos. A maximum entropy model for prepositional phrase attachment. In *Proceedings of the ARPA Human Language Technology Workshop*, pages 250–255, 1994.

[Sal73] A. Salomaa. *Formal Languages*. ACM Monograph Series. Academic Press, New York, 1973.

[Sar98] A. Sarkar. Conditions on Consistency of Probabilistic Tree Adjoining Grammars. In *Proceedings of COLING-ACL '98*, pages 1164–1170, Montreal, 1998.

[Sar01] Anoop Sarkar. Applying co-training methods to statistical parsing. In *Proceedings of NAACL 2001*, Pittsburgh, PA, June 2001.

[SB97] J.-A. S´anchez and J.-M. Bened´ı. 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, September 1997.

[SB98] Wojciech Skut and Thorsten Brants. A maximum-entropy partial parser for unrestricted text. In *Proceedings of the Sixth Workshop on Very Large Corpora*, Montreal, Canada, 1998.

[Sch92] Y. Schabes. Stochastic lexicalized tree-adjoining grammars. In *Proc. of COLING '92*, volume 2, pages 426–432, Nantes, France, 1992.

[Sch00] Sabine Schulte im Walde. Clustering verbs semantically according to their alternation behaviour. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING-2000)*, Saarbrcken, Germany, August 2000.

[Sik97] Klaas Sikkel. *Parsing Schemata*. EATCS Series. Springer-Verlag, 1997.

[Sim00] K. Sima'an. Tree-gram parsing: Lexical dependencies and structural relations. In *Proceedings of the 38th Annual Meeting of the ACL*, pages 53–60, Hong Kong, 2000.

[SJ99] B. Srinivas and Aravind Joshi. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2), 1999.

[SM97] Suzanne Stevenson and Paola Merlo. Lexical structure and parsing complexity. *Language and Cognitive Processes*, 12(2), 1997.

[SM99] Suzanne Stevenson and Paola Merlo. Automatic verb classification using distributions of grammatical features. In *Proceedings of EACL '99*, pages 45–52, Bergen, Norway, 8–12 June 1999.

[SMKW99a] Suzanne Stevenson, Paola Merlo, Natalia Kariaeva, and Kamin Whitehouse. Supervised learning of lexical semantic classes using frequency distributions. In *SIGLEX-99*, 1999.

[SMKW99b] Suzanne Stevenson, Paola Merlo, Natalia Kariaeva, and Kamin Whitehouse. Supervised learning of lexical semantic verb classes using frequency distributions. In *Proceedings of SIGLEX99: Standardizing Lexical Resources*, College Park, Maryland, 1999.

[SN97] J. Stetina and M. Nagao. Corpus based pp attachment ambiguity resolution with a semantic dictionary. In J. Zhou and K. Church, editors, *Proceedings of the Fifth Workshop on Very Large Corpora*, pages 66–80, Beijing and Hong Kong, 1997.

[Sou74] S. Soule. Entropies of probabilistic grammars. *Inf. Control*, 25:55–74, 1974.

[Sri96] B. Srinivas. “Almost Parsing” technique for language modeling. In *Proc. ICSLP ’96*, volume 3, pages 1173–1176, Philadelphia, PA, Oct 3-6 1996.

[Sri97a] B. Srinivas. *Complexity of Lexical Descriptions and its Relevance to Partial Parsing*. PhD thesis, Department of Computer and Information Sciences, University of Pennsylvania, 1997.

[Sri97b] B. Srinivas. *Complexity of Lexical Descriptions: Relevance to Partial Parsing*. PhD thesis, University of Pennsylvania, 1997.

[Sri97c] B. Srinivas. Performance evaluation of supertagging for partial parsing. In *Fifth International Workshop on Parsing Technologies*, Boston, September 1997.

[Sri97d] B. Srinivas. Performance Evaluation of Supertagging for Partial Parsing. In *Proceedings of Fifth International Workshop on Parsing Technology*, Boston, USA, September 1997.

[SS92] Y. Schabes and S. Shieber. An Alternative Conception of Tree-Adjoining Derivation. In *Proceedings of the* 20^{th} *Meeting of the Association for Computational Linguistics*, 1992.

[SS94] Yves Schabes and Stuart Shieber. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124, 1994.

[Sto95] A. Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201, 1995.

[SW95] Yves Schabes and Richard Waters. Tree insertion grammar: A cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics*, 21(4):479–513, 1995.

[SW96] Y. Schabes and R. Waters. Stochastic lexcalized tree-insertion grammar. In H. Bunt and M. Tomita, editors, *Recent Advances in Parsing Technology*, pages 281–294. Kluwer, 1996.

[SZ00] Anoop Sarkar and Daniel Zeman. Automatic extraction of subcategorization frames for czech. In *Proceedings of COLING-2000*, 2000.

[UEGW93] Akira Ushioda, David A. Evans, Ted Gibson, and Alex Waibel. The automatic acquisition of frequencies of verb subcategorization frames from tagged corpora. In B. Boguraev and J. Pustejovsky, editors, *Proceedings of the Workshop on Acquisition of Lexical Knowledge from Text*, pages 95–106, Columbus, OH, 21 June 1993.

[vN94] Gertjan van Noord. Head-corner parsing for TAG. *Computational Intelligence*, 10(4), 1994.

[VS87] K. Vijay-Shanker. *A Study of Tree Adjoining Grammars*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1987.

[Wet80] C. S. Wetherell. Probabilistic languages: A review and some open questions. *Computing Surveys*, 12(4):361–379, 1980.

[WK99] J. Wu and S. Khudanpur. Combining nonlocal, syntactic and n-gram dependencies in language modeling, 1999.

[WM89] Mort Webster and Mitchell Marcus. Automatic acquisition of the lexical frames of verbs from sentence frames. In *Proceedings of the 27th Meeting of the ACL*, pages 177–184, 1989.

[WW89] J. H. Wright and E. N. Wrigley. Probabilistic LR parsing for speech recognition. In *IWPT '89*, pages 105–114, 1989.

[XHPJ01] Fei Xia, Chunghye Han, Martha Palmer, and Aravind Joshi. Automatically Extracting and Comparing Lexicalized Grammars for Different Languages. In *Proc. of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, Seattle, Washington, 2001.

[Xia99] Fei Xia. Extracting tree adjoining grammars from bracketed corpora. In *Proceedings of 5th Natural Language Processing Pacific Rim Symposium (NLPRS-99)*, Beijing, China, 1999.

[Xia01] Fei Xia. *Investigating the Relationship between Grammars and Treebanks for Natural languages*. PhD thesis, University of Pennsylvania, Philadelphia, PA, 2001.

[XPJ00] F. Xia, M. Palmer, and A. Joshi. A Uniform Method of Grammar Extraction and its Applications. In *Proc. of EMNLP/VLC-2000*, 2000.

[XTA01] XTAG Group. A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania, 2001.

[Yar95] David Yarowsky. Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. In *In Proc. 33rd Meeting of the ACL*, pages 189–196, Cambridge, MA, 1995.