The dissertation of Andreas Stolcke is approved:

Chair Date

Date

Date

Date

University of California at Berkeley

1994

Bayesian Learning of Probabilistic Language Models

by

Andreas Stolcke

Bayesian Learning of Probabilistic Language Models

by

Andreas Stolcke

Abstract

The general topic of this thesis is the probabilistic modeling of language, in particular natural language. In probabilistic language modeling, one characterizes the strings of phonemes, words, etc. of a certain domain in terms of a probability distribution over all possible strings within the domain. Probabilistic language modeling has been applied to a wide range of problems in recent years, from the traditional uses in speech recognition to more recent applications in biological sequence modeling.

The main contribution of this thesis is a particular approach to the learning problem for probabilistic language models, known as *Bayesian model merging*. This approach can be characterize as follows.

- Models are built either in batch mode or incrementally from samples, by *incorporating* individual samples into a working model
- A uniform, small number of simple operators works to gradually transform an instance-based model to a generalized model that abstracts from the data.
- Instance-based parts of a model can coexist with generalized ones, depending on the degree of similarity among the observed samples, allowing the model to adapt to non-uniform coverage of the sample space.
- The generalization process is driven and controlled by a uniform, probabilistic metric: the Bayesian posterior probability of a model, integrating both criteria of goodness-of-fit with respect to the data and a notion of model simplicity ('Occam's Razor').

The Bayesian model merging framework is instantiated for three different classes of probabilistic models: Hidden Markov Models (HMMs), stochastic context-free grammars (SCFGs), and simple probabilistic attribute grammars (PAGs). Along with the theoretical background, various applications and case studies are presented, including the induction of multiple-pronunciation word models for speech recognition (with HMMs), data-driven learning of syntactic structures (with SCFGs), and the learning of simple sentence-meaning associations from examples (with PAGs).

Apart from language learning issues, a number of related computational problems involving probabilistic context-free grammars are discussed. A version of Earley's parser is presented that solves the standard problems associated with SCFGs efficiently, including the computation of sentence probabilities and sentence prefix probabilities, finding most likely parses, and the estimation of grammar parameters.

Finally, we describe an algorithm that computes n-gram statistics from a given SCFG, based on solving linear systems derived from the grammar. This method can be an effective tool to transform part of

the probabilistic knowledge from a structured language model into an unstructured low-level form for use in applications such as speech decoding. We show how this problem is just an instance of a larger class of related ones (such as average sentence length or derivation entropy) that are all solvable with the same computational technique.

An introductory chapter tries to present a unified view of the various model types and algorithms found in the literature, as well as issues of model learning and estimation.

Prof. Jerome A. Feldman, Dissertation Chair

Acknowledgments

Life and work in Berkeley and at ICSI for me are marked by a wonderful wealth of friends, colleagues, collaborators, and acquaintances. It will be hard to do justice to the many ways in which the people below, and probably some I will forget to mention, have enriched the years at this extraordinary place—but I will try.

Jerry Feldman, advisor who patiently guides his students by letting them wander about their own ways—a true *Doktorvater*.

Steve Omohundro, collaborator and friend, an endless source of ideas, insight, and above all, enthusiasm.

My extended academic support group, who taught me most of what I know and made Berkeley the place to be (in approximate order of appearance): John Canny, Bob Wilensky, Stuart Russell, George Lakoff, Chuck Fillmore, Lotfi Zadeh, Jitendra Malik, John Ousterhout, and David Culler.

Peter Cheeseman and Wray Buntine, who introduced me to the powers of Bayes' Rule at just the right moment and have been a willing source of advice ever since—may their book be on the shelves soon.

Nelson Morgan, who adopted me into his group, and continues to be a valuable source of advice about the real data, and RAP.

Subutai Ahmad, Dekai Wu, Terry Regier and Ben Gomes, officemates, friends and collaborators: thanks for making these such fun years.

The ICSI AI and realization groups, Bertrand Irissou, Brian Kingsbury, Chris Bregler, Chuck Wooters, Dan Jurafsky, David Bailey, David Stoutamire, Eric Fosler, Gary Tajchman, Hervé Bourlard, Jeff Bilmes, Joachim Diederich, Jonathan Segal, Krste Asanovic, Lokendra Shastri, Nikki Mirghafori, Srini Narayanan, Steve Greenberg, Susan Weber, and Yochai Konig. Apart from being an utterly fun crowd, several of them did much of the work that was crucial in obtaining results.

All of the ICSI staff, past and present, who have developed perfection in providing unobtrusive support and afternoon tea and cookies. Special thanks go to Alix Collison for friendship and help in feeling, quite literally, at home.

The rest of the Berkeley AI and Cogsci crowd, Adele Goldberg, Francesca Barrientos, Jane Edwards, Marti Hearst, Mike Braverman, Mike Schiff, NJ, and Peter Norvig—never short of advice, encouragement, good humor, rhythm, and blue notes.

Kathryn Crabtree, who has been indispensable in steering me (as well as everyone else) through, and where possible clear of the thicket of University regulations and requirements.

My former academic advisors at Munich, Erwin Kloeck, Christian Freksa, Wilfried Brauer, and Theo Vennemann, who in many ways laid the foundations for the present work and encouraged me to try my luck in Berkeley.

Fernando Pereira, who I first ran into on the Internet, and who has since made himself indispensable with valuable insights and bibliographic references.

Our 'Foster' parents, Bill and Donna, and everybody at WCPC, who made us feel so welcome in the very beginning and thereafter.

David Blake, Berkeley student *par excellence* and important source of information about American and Berkeley culture.

My parents, whose support, encouragement and understanding I could always count on, and who (as they say) made it all possible.

Susanne, Patrick and Oliver, who made everything bearable, even though it wasn't always for them. These pages are dedicated to you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

The general topic of this thesis is the probabilistic modeling of language, in particular natural language. In probabilistic language modeling, one tries to characterize the strings of phonemes, words, etc. of a certain domain in terms of a probability distribution over all possible strings within the domain. The language and mathematical framework of probability theory is used to formalize questions such as

- Given a language, which of two string is more likely to be observed?
- Given a language and a string, which of two derivations (or analyses) of the string is more likely?
- Given two languages, how similar (or different) are they?
- Given a language described in some way and a corpus of samples, how good a description of the data is the language?

All of these problems have numerous applications, and the success of the probabilistic approach is reflected by the large number and the diversity of the domains in which probabilistic language models have been applied to advantage in recent years. These include both established applications such as speech recognition and understanding, as well a new ones, such as biological sequence modeling.

A prerequisite for most of these applications is the *learning problem*: given a sample corpus that is assumed to be representative of the domain, find an adequate probabilistic description of it. The main topic of this thesis is a particular approach to the learning problem for probabilistic language models, with the following characteristic features:

- Models are built either in batch mode or incrementally from samples, by *incorporating* individual samples into a working model
- A uniform, small number of simple operators works to gradually transform an instance-based model to a generalized model that abstracts from the data.

- Instance-based parts of a model can coexist with generalized ones, depending on the degree of similarity among the observed samples, allowing the model to adapt to non-uniform coverage of the sample space.
- The generalization process is driven and controlled by a uniform, probabilistic metric: the Bayesian posterior probability of a model, integrating both criteria of goodness-of-fit with respect to the data and a notion of model simplicity ('Occam's Razor').

Our[1] approach is quite general in nature and scope (comparable to, say, Mitchell's version spaces (Mitchell 1982)) and needs to be instantiated in concrete domains to study its utility and practicality. We will do that with three different types of probabilistic models: Hidden Markov Models (HMMs), stochastic context-free grammars (SCFGs), and simple probabilistic attribute grammars (PAGs).

Following this introduction, Chapter 2 presents the basic concepts and mathematical formalisms underlying probabilistic language models and Bayesian learning, and also introduces our approach to learning in general terms.

Chapter 3 (HMMs), Chapter 4 (SCFGs) and Chapter 5 (attribute grammars) describe the particular versions of the learning approach for the various types of languages models. Unfortunately, these chapters (except for Chapter 3) are not entirely self-contained, as they form a natural progression in both ideas and formalisms presented.

The following two chapters address various computational problems aside from learning that arise in connection with probabilistic context-free language models. Chapter 6 deals with probabilistic parsing and Chapter 7 gives an algorithm for approximating context-free grammars with much simpler n-gram models. These two chapters are nearly self-contained and need not be read in any particular order (with respect each other or the preceding chapters).

Chapter 8 discusses general open issues arising from the present work and gives an outlook on future research.

Virtually all probabilistic grammar types and algorithms described in the following chapters have been implemented and integrated in an object-oriented framework in CommonLisp/CLOS. The result purports to be a flexible and extensible environment for experimentation with probabilistic language models. The documentation for this system is available separately (Stolcke 1994).

The remainder of this introduction gives general motivation and highlights, as well as some historical background.

1.2 Structural Learning of Probabilistic Grammars

Probabilistic language models (or grammars) have firmly established themselves in a number of areas in recent time (automatic speech recognition being one of the major applications). One important factor is their probabilistic nature itself: they can be used to make weighted predictions about future data

[1]The first person plural will be used throughout, both for stylistic uniformity and to reflect the fact that much of this work was done in collaboration with others. Bibliographic references to co-authored publications are given at the end of this chapter.

based (or conditioned) on evidence seen in the past, using the framework of probability theory as a consistent mathematical basis.

However, this fundamental feature is only truly useful because probabilistic models are also *adaptable*: there are effective algorithms for tuning a model based on previously observed data, so as to optimize its predictions on new data (assuming old and new data obey the same statistics).

1.2.1 Probabilistic finite-state models

An example in point are the probabilistic finite-state models known as *Hidden Markov models (HMMs)* routinely used in speech recognition to model the phone sequences making up the words to be recognized. The top part of Figure 1.1 shows a simple word model for "and." Each phonetic realization of the word corresponds to a path through the network of states and transitions, with the probabilities indicated. Given a network structure and a training corpus data to be modeled, there are standard algorithms for optimizing (or estimating) the probability parameters of the HMM to fit the data.

However, a more fundamental problem is how to obtain a suitable model structure in first place. The basic idea here will be to construct initial model networks from the observed data (as shown in the bottom part of Figure 1.1), and then gradually transform them into a more compact and general form by a process called *model merging*. We will see that there is a fundamental tension between optimizing the fit of the model to the observed data, and the goal of generalizing to new data. We will use the Bayesian notions of prior and posterior model probabilities to formalize these conflicting goals and derive a combined criterion that allows finding a compromise between them.

Chapter 3 describes this approach to structural learning of HMMs and discusses many of the issues and methods recurring in later chapters.

1.2.2 The Miniature Language Learning (L_0) Task

An additional motivation for the present work came from a seemingly simple task proposed by Feldman *et al.* (1990): construct a machine learner that could generalize from usage examples of a natural language fragment to novel instances, for an arbitrary natural language. Figure 1.2 shows the essential elements of this *miniature language learning* problem, informally known as the "L_0" task. The goal is to 'learn' the L_0 language from exposure to pairs of corresponding two-dimensional pictures and natural language descriptions. Both the syntax and semantics of the language were intentionally limited to make the problem more manageable. The purpose of the proposal was to highlight certain fundamental problems with traditional cognitive science theories, including issue such as dependence on the underlying conceptual system, grounding of meaning and categorization in perception, and others which are explored in recent and ongoing research (Regier 1992; Feldman *et al.* 1994).

For our purposes we can abstract a (much simpler) subproblem from this interdisciplinary task: given pairs of sentences and associated idealized semantics (e.g., in first-order logic formulae), construct an adequate formal description of the relation between these two for the given language.

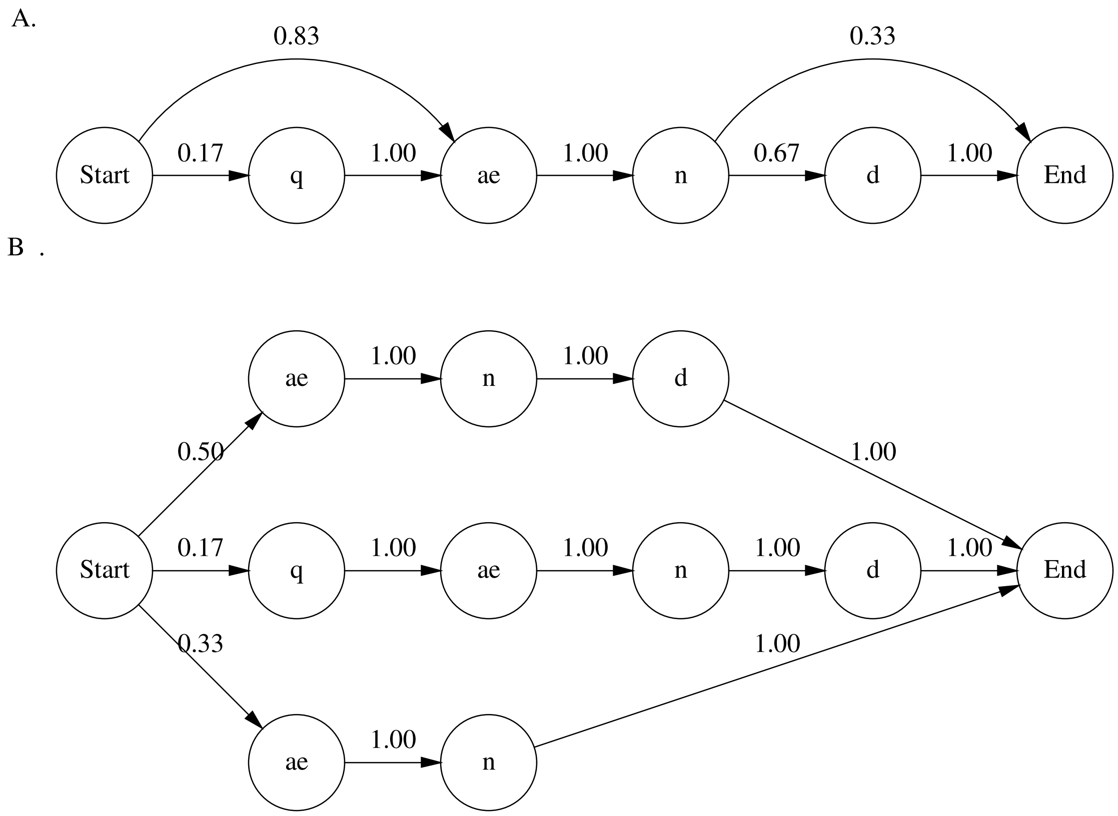
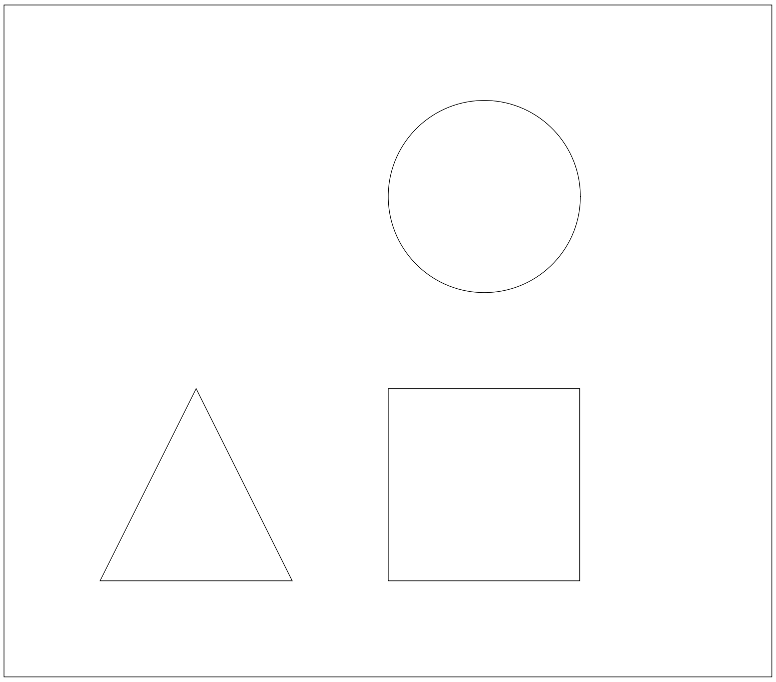


Figure 1.1: Hidden Markov Models for the word "and"

A. Generalized word model derived by model merging. B. Initial model derived from observed data, prior to model merging. The symbol "q" represents a glottal stop.

Chapters 4 and 5 can be viewed as a study on how to bridge the gap between this formal syntax/semantics learning problem and the probabilistic methods that were originally developed for the more constrained finite-state models mentioned earlier. To this end, we first relax the finite-state constraint on the syntactic part of the learning problem, and move to probabilistic *context-free* models (Chapter 4). The result is an extended model merging algorithm that compares favorably with several alternative computational context-free learning approaches proposed in the literature, while coping with a significant portion of the syntactic structures found in the L_0 task.

The final extension of the model merging approach is designed to allow the combined learning of miniature language syntax and semantics (Chapter 5). The basic idea here is to capture the correspondence between language elements and their meaning by attaching *probabilistic attributes* to the nonterminals of a (likewise probabilistic) context-free grammar. This approach turns out to impose rather strong restrictions, for both learning and the expressive power of the model, but is nevertheless sufficient to learn a basic version of



"the circle is above a square"

"the triangle is to the left of the square"

Figure 1.2: The L_0 task

The learner is exposed to picture-sentence pairs as shown here (in fixed, but arbitrary natural language), and has to generalize so as to be able to determine the truth of novel picture-sentence pairs. The baseline version of this task uses simple static binary spatial relations and correspondingly limited syntax as shown here. Various variants that extend the scope of the task are given in Feldman *et al.* (1990).

the L_0 language. While the resulting system cannot do full justice to the original L_0 problem, it still provides a useful example of both the possibilities and the limitations of the model merging approach.

1.3 Miscellaneous topics

In the second part of this thesis we discuss various computational issues which are related to, but not directly part of the learning problem for probabilistic language models. The focus here is on *stochastic context-free grammars (SCFGs)*, which, in spite of their shortcoming, represent the current state-of-the-art in computational approaches to probabilistic natural language processing. Historically, the algorithms presented here were developed to meet specific needs, either as part of the implementation of the learning algorithms described in the first part, or as a result of the tie-ins of the present work with an ongoing project in speech understanding. (Jurafsky *et al.* 1994a).

Chapter 6 describes a parser for SCFGs that solves several of the standard tasks for probabilistic grammars in a single elegant and efficient framework, namely a probabilistic generalization of Earley's algorithm for regular CFGs. In particular, we show how fairly straightforward extensions to Earley's parser allow the computation of sentence probabilities, most likely parses, and the estimation of the grammar parameters. The algorithm is of special interest because it also allows efficient, incremental computation of the probabilities of possible next words for prefixes of sentences generated by a SCFG.

Chapter 7 solves a related problem: given a probabilistic context-free description of a language, compute low-level statistics of the type: "what is the probability of the next word being X given that the previous two words were Y and Z?" Such statistics, known as *n-gram probabilities*, are essential to a number of standard language modeling applications which cannot easily draw directly on higher-level models (such a SCFGs) for computational reasons. For these, the algorithm is a useful tool for transfering some of the higher-level probabilistic structure into the lower-level representation. The appendix of Chapter 7 also surveys a number of related computational tasks for SCFGs (such as computing the average sentence length) that are solvable in a manner similar to the n-gram problem.

1.4 Bibliographical Note

Various portions of this thesis are based on work published previously, listed here for reference: Chapter 3 (Stolcke & Omohundro 1993; Stolcke & Omohundro 1994a; Wooters & Stolcke 1994), Chapter 4 (Stolcke & Omohundro 1994b), Chapter 6 (Stolcke 1993), Chapter 7 (Stolcke & Segal 1994), (Jurafsky *et al.* 1994b). Any remaining errors are of course the sole responsibility of this author.

Chapter 2

Foundations

2.1 Preliminaries

In this chapter we review some basic definitions and concepts that will be used throughout the following chapters. We also introduce the classes of probabilistic language models that are the subject of this thesis,[1] along with the relevant standard algorithms.

The range of probabilistic models and comcepts covered here is slightly wider than strictly necessary for the rest of the thesis, in order to point out some generalizations and connections between various approaches that are often not found in the specialized literature.

2.2 Probabilistic Language Models

A *probabilistic language model* or *probabilistic grammar* generalizes the classical notion of a grammar as an acceptor/rejector of strings to the probabilistic domain. Specifically, the *(probabilistic) language* generated by such a grammar is a probability distribution over strings, written

$$P(X|L)$$

where X is a random variable ranging over strings, and L denotes the language in question. The notation is that of a conditional distribution, as we can think of a particular probability $P(x|L)$ as the conditional probability of drawing the string x given that we have previously fixed the language to be L. (This suggests that the language itself is drawn probabilistically from a pool of possible languages; this is indeed a crucial ingredient of the approach developed later on.) The domain of the probability distribution are the finite strings over a fixed, discrete alphabet Σ.

A probabilistic grammar M (as in model) is a description of such a distribution. The language generated by a model M is $L(M)$, but when writing probabilities and distributions we usually collapse the

[1] This is includes all the 'standard' models; probabilistic attribute grammars will be deferred until Chapter 5.

distinction between language (distribution) and grammar (description), and simply write

$$P(X|M) = P(X|L(M))$$

2.2.1 Interpretation of probabilities

The probabilities assigned to strings by a model can be interpreted as long-term relative frequencies of those strings, assuming conditional independence of samples. This *frequentist interpretation* becomes problematic as we introduce probabilities of entities that do not correspond to outcomes of repeatable experiments, such as the models themselves.

In such cases it is more plausible to think of probabilities as *degrees of belief*, the *subjectivist interpretation* of probabilities. The classic paper by Cox (1946) shows that the two interpretations are compatible in that they observe the same calculus if some simple assumptions about beliefs and their compositional properties are made.

The unification of the frequentist and the subjectivist treatment of probabilities is fundamental to the Bayesian approach described later. It allows probabilities to be used as the common 'currency' relating observations and beliefs about underlying explanations for observations.

2.2.2 An example: n-gram models

A simple example both illustrates these notions and introduces a useful standard tool for later use. An n-*gram model* defines the probability of a string $P(x|M)$ as a product of conditional probabilities

$$\begin{aligned} P(x|M) &= P(x_1|\$; M)P(x_2|\$x_1; M)\ldots P(x_n|x_1\ldots x_{n-1}; M) \\ &\quad \ldots P(x_i|x_{i-n+1}\ldots x_{i-1}; M) \\ &\quad \ldots P(x_l|x_{l-n+1}\ldots x_{l-1}; M)P(\$|x_{l-n+2}\ldots x_l; M) \quad , \end{aligned} \tag{2.1}$$

where l is the string length, x_i is the ith symbol in string x, and \$ is a special delimiter marking beginning and end of a string.

The parameters of an n-gram model are thus the probabilities

$$P(x_n|x_1\ldots x_{n-1})$$

for all $x_1, \ldots, x_n \in \Sigma \cup \{\$\}$. (It is convenient to allow a prefix of $x_1 \ldots x_{n-1}$ to take on the value \$ to denote the *left end* of the string. The context always makes it clear which end of a string \$ stands for.)

It is useful to compare definition (2.1) to the expression for $P(x|M)$ arrived at by repeated conditioning, which is true of any probability distribution over strings:

$$P(x|M) = P(x_1|\$)P(x_2|\$x_1)\ldots P(x_l|\$x_1\ldots x_{l-1})P(\$|\$x_1\ldots x_l) \tag{2.2}$$

We see that that n-gram models represent exactly those distributions in which each symbol is independent of the beginning of the string given just the immediately preceding $n-1$ symbols (including the position of the left string boundary).

The number of parameters in n-gram models grows exponentially with n, and only the cases $n = 2$ (*bigram models*) and $n = 3$ (*trigram models*) are of practical importance. Bigram and trigram models are popular for various applications, especially speech decoding (Ney 1984), to *approximate* the *true* distributions of language elements (characters, words, etc.), which are known to violate the independence assumption embodied in (2.1).

Because (2.1) is essentially a truncated version of the true joint probability given by (2.2), n-grams are in some sense a natural choice for this approximation, and are appropriate to the extent that symbol occurrences tend be more and more independent as the distance between the occurrences increases. Of course there are important cases in natural language and elsewhere where this assumption is blatantly wrong. For example, the distribution of lexical elements in natural languages are constrained by phrase structures that can relate two (or more) words over essentially arbitrary distances. This is the main motivation for moving, at a minimum, to the stochastic context-free models that are one of the main subjects of this thesis.

2.2.3 Probabilistic grammars as random string generators

One can always abstractly associate a probabilistic language L with a corresponding random string generator, i.e., a device that generates strings stochastically according to the distribution L. However, a probabilistic grammar usually describes L by making this generator concrete. For example, for n-grams a generator would output string symbols left-to-right, always choosing the next symbol x_i according to a probability lookup table indexed by the $(n-1)$-tuple of previous symbols $x_{i-n+1} \dots x_{i-1}$. One possible choice is \$, which causes the generator to stop.

The probability of a string is thus equivalent to the joint probability of a sequence of roles of dice, each die corresponding to the conditional distribution of a certain $n-1$-gram. (Hence there are roughly $|\Sigma|^{n-1}$ dice.)[2] Each die has $|\Sigma| + 1$ faces, one for each possible next symbol or \$.

In a sense, n-gram models represent a 'brute force' approach to probabilistic language modeling, and are linguistically not very interesting due to their inability to describe complex structures and non-local dependencies among language elements. However, in many applications they give surprisingly good results that are not easy to improve upon with more sophisticated models. For this reason, and because they can be processed very efficiently they have come to be the method of choice in tasks such a speech decoding. In Chapter 7 we will revisit n-gram models and describe an algorithm for deriving them from more complex probabilistic models, rather than estimating them directly from data.

2.2.4 Multinomial distributions

All the language models considered in this thesis can be described as generating strings through a sequence of (generalized) die tosses. It is therefore useful to review some basic properties of *multinomial distributions*, the class of distributions describing the outcomes of generalized dice.

[2] The actual number is $\frac{|\Sigma|^n - 1}{|\Sigma| - 1}$, taking into account the left contexts containing the \$ marker.

A multinomial distribution (*multinomial*, for short) of dimension n is specified by a tuple of parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$, $0 \leq \theta_i \leq 1$, $\sum_{i=1}^{n} \theta_i = 1$. The multinomial distribution itself describes the probability of the possible outcomes of c independent samples drawn according to $\boldsymbol{\theta}$. Since that probability depends only on the counts for each outcome, $c_1, \ldots, c_n$, we can write

$$P(c_1, \ldots, c_n | c, \boldsymbol{\theta}) = \frac{c!}{c_1! \cdots c_n!} \prod_{i=1}^{n} \theta_i^{c_i} \quad . \tag{2.3}$$

This is the distribution for *unordered samples*, where sequences of outcomes that are permutations on one another are considered to be the same joint event. For ordered samples the distribution is simply

$$P(c_1, \ldots, c_n | c, \boldsymbol{\theta}) = \prod_{i=1}^{n} \theta_i^{c_i} \quad , \tag{2.4}$$

i.e., the multinomial coefficient can be dropped.

The usual scenario throughout this thesis is that the outcomes (samples) are given, and one wants to draw conclusions about $\boldsymbol{\theta}$. In this case the difference between (2.3) and (2.4) is a constant, and can safely be ignored. We will therefore use mainly the ordered sample scenario since it is described by the simpler formula (2.4).

The following formulas summarize the means (expectations), variances, and covariances, respectively, for the multinomial.

$$E(c_i) = c\theta_i \tag{2.5}$$

$$\text{Var}(c_i) = c\theta_i(1 - \theta_i) \tag{2.6}$$

$$\text{Cov}(c_i, c_j) = -c\theta_i\theta_j \tag{2.7}$$

2.2.5 Parameter estimation

A fundamental problem for probabilistic language modeling is the inference of parameter values (such as $\boldsymbol{\theta}$ vectors in an n-gram grammar) from available data. If the parameters fully describe the choice of a the model then parameter estimation subsumes the 'learning' problem for grammars of the given class. Although it is possible to parameterize all aspects of a grammar, we will see later that is often useful to draw a distinction between parameters of the continuous type found in n-grams and other, 'structural' aspects of a model.

In the language of statistics, an *estimator* $\hat{\theta}$ for a parameter θ is simply a function that computes a putative value for θ given a set of samples X: $\hat{\theta} = \hat{\theta}(X)$. The intuitive requirement that $\hat{\theta}$ should be 'close' to the actual value θ is formalized by the notions of *bias* and *consistency*. An estimator is *unbiased* if its expected value given data drawn from the distribution specified by θ is θ itself: $E(\hat{\theta}(X)|\theta) = \theta)$. Otherwise one can quantify the *bias* as $E(\hat{\theta}(X)|\theta) - \theta$. An estimator is $consistent$ if it converges to the true θ as amount of data increases.

For example, for multinomials an unbiased, consistent estimator of the component probabilities θ_i is given by the observed averages

$$\hat{\theta}_i = \frac{c_i}{c} \quad . \tag{2.8}$$

Zero bias is not always desirable, however, as attested by the extensive literature on how to modify estimates for n-gram (and similar) models so as to avoid estimates from taking on 0 values, specifically in linguistic domains. According to (2.8) this would be the case whenever some outcome has not been observed at all ($c_i = 0$). In many domains and for realistic sample sizes one expects many counts to remain zero even though the underlying parameters are not; hence one wants to bias the estimates away from zero. See Church & Gale (1991) for a survey and comparison of various methods for doing that. Another reason for introducing bias is to reduce the variance of an estimator (Geman *et al.* 1992).

2.2.6 Likelihood and cross-entropy

If the data X is given and fixed, we can view $P(X|M)$ as a function of M, the *likelihood (function)*. A large class of estimators, so called *maximum likelihood (ML) estimators*, can be defined as the maxima of likelihood functions. For example, the simple estimator (2.8) for multinomials happens to also be the ML estimator, as setting the parameters θ_i to their empirical averages maximizes the probability of the *observed* data.

Intuitively, a high likelihood means that the model M 'fits' the data well. This is because a model allocates a fixed probability mass (unity) over the space of possible strings (or sequences of strings). To maximize the probability of the observed strings the unobserved ones have as little probability as possible, within the constraints of the model. In fact, if a model class allows assigning probabilities to samples according to their relative frequencies as in (2.8), this is the best one can do.

An alternative measure of the fit or closeness of a model to a distribution is based on the concept of entropy. The *relative entropy* (also known as the Kullback-Leibler distance) between two distributions p and q is defined as

$$D(p||q) = -\sum_x p(x) \log \frac{q(x)}{p(x)} \tag{2.9}$$

This can be written as

$$D(p||q) = -\sum_x p(x) \log q(x) - \sum_x p(x) \log p(x)$$

The second sum is the familiar entropy $H(p)$ of the distribution p, whereas the first sum is an entropy-like term in which both distributions appear. We will call this first term the *cross-entropy*[3] $H_p(q)$, giving

$$D(p||q) = H_p(q) + H(p) \quad .$$

[3] In the literature the terms relative entropy an cross-entropy are often used synonymously, to refer to both $D(p||q)$ and $H_p(q)$. The probable reason is that for minimization either can be used (see below). We find it both useful and intuitive to make the distinction as done here—think of $H_p(q)$ as a 'cross-over' between $H(p)$ and $H(q)$.

It can be shown that $H_p(q) \geq H(p)$ always, with equality if and only if the two distributions are identical. It follows that $D(p||q) \geq 0$, with $D(p||q) = 0$ iff $p = q$. This justifies thinking of $D(p||q)$ as a pseudo-distance between distributions, or between distributions and models.[4]

Computing $D(p||q)$ exactly presumes knowledge of the full distributions p and q, but in typical scenarios at most one is known, e.g., because it is given by a model. For example, we might use the relative entropy to define an estimator for model parameters, such that the estimated value is that which *minimizes* $D(p||L(M))$, where p is the distribution from which the samples are drawn. Since p is not known, we cannot compute D directly. However, note that for minimization purposes only the $H_p(q)$ term is relevant, $H(p)$ is an unknown constant, but one that remains fixed as M is varied. This leaves $H_p(q)$ to be minimized, which is the expected value of $-\log(q)$ under the true distribution $p(x)$. It can therefore be estimated by averaging over the sample corpus X

$$H_p(p) \approx -\frac{1}{|X|} \sum_{x \in X} \log q(x) \tag{2.10}$$

We thus see that estimated cross-entropy is proportional (by a factor of $-\frac{1}{|X|}$) to the log of the likelihood. Therefore, ML estimators are effectively also minimum relative entropy estimators.

2.3 Grammars with hidden variables

Although all grammars considered here generate their samples through a combination of multinomials, the sequence of choices that can give rise to a given sample is not always uniquely determined, unlike for n-gram grammars. There, one can uniquely identify the sequence of choices leading to the generation of a complete string, by inspecting the n-grams occurring in the string in left-to-right order. Knowing the n-grams, one can then compute their probabilities, and hence the probability of the string itself (by taking products).

A complete sequence of generator events (multinomial samples) that generate string x is called a *derivation* of x. (Thus, for n-gram models the only derivation of a strings is the string itself.) Grammars that generate strings with more than one derivation are called *ambiguous*. Each derivation d has a probability which is the product of the probabilities of the multinomial outcomes making up the derivation. In general, then, a string probability is a sum of derivation probabilities, namely, of all derivations generating the same string:

$$P(x|M) = \sum_{d \text{ derives } x} P(d|M)$$

We can think of a derivation (or parts thereof, if it can be conveniently decomposed), as an unobserved, 'hidden' random variable.

[4]However, relative entropy is not symmetric and does not obey the triangle inequality.

2.3.1 Mixture grammars

A simple example for this are grammars that generate strings from two subsidiary, component grammars, M_1 and M_2. The first step in a derivation is choosing which of the two submodels is to generate the string, according to a single mixture probability μ. Following that, the chosen submodel generates the string according to whatever distribution it represents. The total probability of x is thus

$$P(x|\mu, M_1, M_2) = \mu P(x|M_1) + (1 - \mu)P(X|M_2) \tag{2.11}$$

or a weighted mixture of the component distributions. This type of model can obviously be generalized to any number of components (including non-finite mixtures). It is in a sense the probabilistic version of a *union* of formal languages. The key feature in this context is that a string x that can be generated by both M_1 and M_2 has at least two possible derivations, whose identity is not observable.

Mixing models is also known in the language modeling literature as *interpolation* (Bahl *et al.* 1983). The term is most often used when the mixture proportions are estimated from independent training data, rather than using the EM algorithm, described next.[5]

2.3.2 Expectation-Maximization

Hidden variables (or ambiguity) in grammars create an additional problem for parameter estimation as the observed outcomes for the multinomials in the grammar become ambiguous as well, and can no longer be simply counted for use with estimators like (2.8).

In many cases the estimation in the face of hidden variables can be solved using a general technique called *expectation-maximization* (Dempster *et al.* 1977), or *EM*. Intuitively, since the likelihood is the objective function to be maximized, but some supposedly known terms in it are not available (e.g., the string derivations), one proceeds by 'guessing' the values of the unobserved hidden variables (the *E-step*), and then maximizes the likelihood over the parameters given both the guessed values and the known observables (the *M-step*). Since the reestimation may affect the guessing process one iterates the two steps until the procedure converges to a set of parameter values. The guessed values are typically the *expectations of the unknown statistics* needed to compute the likelihoods, hence the name of the E-step.[6] It can be shown that on each EM iteration the likelihood is non-decreasing, thus converging to a local maximum in the parameter space. This leaves open whether local likelihood maximization is good enough in practice, a question we will return to.

[5] The term *deleted interpolation* refers to the holding-out method used in estimating the mixture proportions.

[6] This is a simplified characterization. In full generality, one maximizes the expectation of the log likelihood function, i.e., the E-step consists of determining the function

$$L(\theta') = E_{P(d|x,\theta)} \log P(x, d|\theta'),$$

where θ are the fixed current parameters, d the hidden variables, and x the observed variables. For many standard distributions (in particular, those of the exponential family), the expected log likelihood is the log likelihood assuming the sufficient statistics take on their expected values. For example, for the multinomial:

$$E \log P(x, d|\theta) = E \sum_i c_i \log \theta_i = \sum_i (E c_i) \log \theta_i \quad .$$

EM for mixtures As a brief illustration of the EM method we consider the case of the simple mixture model (2.11) (Redner & Walker 1984). In the E-step we compute the expected values for the counts c_i, the number of times submodel i was used to generate a string. Since the samples are independent, this can be done by summing fractional expected counts, i.e., posterior probabilities for the samples coming from each of the mixture components:

$$Ec_i = \sum_{x \in X} P(d_i|x, M) \tag{2.12}$$

where $d_i = 1$ iff submodel M_i was chosen to generate x, and $d_i = 0$ otherwise. This posterior probability of a submodel generating x can be computed using Bayes' Law:

$$\begin{aligned} P(d_i|x, M) &= \frac{P(d_i|M)P(x|d_i, M)}{P(x|M)} \\ &= \frac{\mu_i P(x|M_i)}{P(x|M)} \end{aligned}$$

In the two-component case, $\mu_1 = \mu$ and $\mu_2 = 1 - \mu$.

We obtain the intuitive result that the 'expected attribution' of a sample x to one of the submodels M_i is proportional to how well that submodel 'explains' the sample, measured by its likelihood $P(x|M_i)$, weighted by the prior probability of choosing M_i.

2.3.3 Viterbi derivations and approximate EM

Often one is interested in the *single most likely* derivation for a string x within a model M,

$$\operatorname*{argmax}_d P(d|x, M)$$

This is called the *Viterbi* derivation of x, and is some sense the 'best' explanation of x within M (Viterbi 1967). Finding the Viterbi derivation is the canonical way to disambiguate a string x using probabilistic grammar M.

Another important use of Viterbi derivations is in approximating the full probability of a string, which is obtained by summing over all its derivations:

$$P(x|M) = \sum_d P(x|d, M)P(d|M) \approx \max_d P(x|d, M)P(d|M)$$

By definition, the maximum in the approximate term is achieved by choosing d to be the Viterbi derivation.

The Viterbi approximation of the likelihood is also frequently used in simplifying the EM algorithm. Instead of maximizing with respect to the true expectations of hidden variables, one approximates the expectations by counting as if the Viterbi derivation had occured with probability one. For example, to estimate the mixture proportion in (2.11) using Viterbi, one would replace Ec_i by the number of times submodel M_i was the *a posteriori* more probable of the two.[7]

[7] If the submodels themselves are ambiguous then this leaves open two possibilities: either one counts the most probable joint derivations (component choice + most likely derivation within the component), or simply the most probable component choice using the full submodel likelihoods (summing over all derivation within the submodels). The latter is the better approximation, but the former may be more practical to compute.

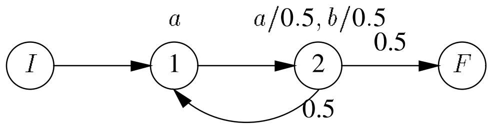


Figure 2.1: Simple HMM.

State names appear within circles, outputs above their emitting states.

2.3.4 Hidden Markov Models

Hidden Markov Models (HMMs) are the probabilistic counterpart of nondeterministic finite automata (NFAs) (Hopcroft & Ullman 1979). As NFAs, HMMs have a finite number of states, including the initial and final ones. States are connected by transitions and emit output symbols from a finite, discrete alphabet. (HMMs can also be defined to emit output symbols on the transitions rather than at states, as is usual with NFAs. However, the two variants are equivalent and can be converted into each other.) As NFAs, HMMs generate (or accept) strings over the output alphabet by nondeterministic walks between the initial and final states. In addition, HMMs also assign probabilities to the strings they generate, computed from the probabilities of individual transitions and emissions. A sequence of states, or *path*, leading from initial to final state constitutes an HMM *derivation* in the sense introduced above.

HMMs get their name from the the fact that the underlying state sequence is the result of a Markov process, which is 'hidden' from observation by a corresponding, but non-unique sequence of emissions. We will be discussing only first-order HMMs where state dependencies are on the immediate predecessor only; higher-order HMMs can have states depending on a limited number of predecessors. Higher-order HMMs may be transformed into equivalent first-order ones.

HMM are also used in the modeling of unbounded strings, in which case there are no final states. Instead, under certain reachability conditions, the model settles into a unique stationary distribution of states and outputs (*e.g.*, Cover & Thomas (1991)). Also, there is no reason why the model is limited to discrete outputs. Continuous output distributions are commonly used in modeling speech, for example. However, we will restrict ourselves to finite string generation and discrete outputs.

A formal definition of HMMs will be deferred to Chapter 3. Instead, we will illustrate the basic notions by way of an example (Figure 2.1).

The model generates strings from the regular language $(a(a \cup b))^*$. All transition and emission probabilities are 1 except where labeled otherwise. The exceptions are at state 2, which outputs a or b with probability 0.5 each, and where transitions occur to either state 1 or the final state F, again with probability 0.5.

Being a probabilistic model, it assigns probabilities to strings according to possible paths through the model. A *path* is a sequence of states from initial to final state, and is the HMM version of a derivation, in

```
S --> a         0.2
  --> b         0.2
  -->           0.1
  --> a S a     0.4
  --> b S b     0.1
```

Figure 2.2: Simple stochastic context-free grammar generating palindromes

.

the general sense introduced earlier. $P(x|M)$ is the sum of all the joint probabilities of random walks through the model (i.e., derivations) generating x.

In the example HMM, the string $abaa$ is generated by only a single path through the model, and hence

$$\begin{aligned} P(abaa|M) &= p(q_I \to q_1)p(q_1 \uparrow a)p(q_1 \to q_2)p(q_2 \uparrow b) \\ & p(q_2 \to q_1)p(q_1 \uparrow a)p(q_1 \to q_2)p(q_2 \uparrow a)p(q_2 \to q_F) \\ &= (0.5)^4 = 0.1375. \end{aligned}$$

The conditional probabilities in the product are the individual transition and emission probabilities. The fact that each is conditional only on the current state reflects the Markovian character of the generation process.

2.3.5 Stochastic Context-free Grammars

Stochastic context-free grammars (SCFGs) are the natural extension of context-free grammars (CFGs) to the probabilistic realm. We again present a simple example and leave the formal presentation to Chapter 4.

Each context-free production is annotated with the conditional probability that it is chosen among all possible productions when the left-hand side nonterminal is expanded. Figure 2.2 shows a SCFG generating all the palindromes over the alphabet $\{a, b\}$. A derivation in this case is the usual tree-structure (parse tree) arising from the nonterminal expansions, and its probability is the product of the probabilities of the rules involves. Hence, the string `abbba` is generated with probability $0.4 \times 0.1 \times 0.1 \times 0.2$, corresponding to the only derivation of this string in the grammar.

Notice how the notion of context-freeness has been extended to include *probabilistic conditional independence* of the expansion of a nonterminal from its surrounding context. This turns out to be crucial to keep the computational properties of SCFGs reasonable, but it is also the major drawback of SCFGs when using them to model, say, natural language. We will return to this problem at various points in the following chapters.

2.4 Levels of Learning and Model Merging

2.4.1 Beyond parameter estimation

Learning of probabilistic language models from data involves the choice of a model that best represents the distribution that generated the samples. We can identify three levels at which choices are involved:

a) Choice of a model class (n-gram, finite-state, context-free, mixture of several of these, etc.).

b) Choice of a model structure (how many states, nonterminals, what productions, etc.)

c) Choice of parameters (n-gram probabilities, transition probabilities, etc.)

So far we have briefly addressed (c), with maximum likelihood and EM estimators. These are in fact the established methods for all of the language models discussed in this thesis. The main contribution of the thesis will be to (b), the choice of model structure. Level (a) is currently left to the human designer of a learning system. The fundamental problem of model design will be briefly discussed in in Chapter 8.

The boundaries between (a) and (b), as well as between (b) and (c), are inherently fuzzy, due to *subsumption* between representations. For example, in a finite-state model the presence or absence of transitions can be represented either as part of the model structure (level b) or a continuous transition probability parameter (level c), where probability zero effectively represents the absence of a transition. These two representation correspond to very different learning approaches: to learn (b) the learner has to make discrete decisions, whereas for (c) learning typically is accomplished by gradual adjustment of the continuous parameters (using EM, gradient ascent, etc.).

Similarly, when faced with a choice between different model classes (level a) it is often possible to devise a generalized class that subsumes the alternatives in question. For example, the class of probabilistic context-free models contains finite-state models as a special case. One can then hope that whatever structure learning method is used (level b) will pick out the right type of model. In practice this is not quite so straightforward since one might end up with a model structure that doesn't fall neatly into any of the target classes.

2.4.2 Model merging

The approach to the structural learning problem for probabilistic grammars pursued in this thesis is conceptually based on a rather general method recently advocated by Omohundro (1992) called *model merging*. Many of the key ingredients of this method are very old and can be found in algorithms from several disparate areas dealing with various incarnations of the data modeling task. Omohundro argues that there are domain-independent principles at work that lend the model merging approach generality, efficiency and a certain degree of cognitive plausibility.

The basic idea is that a model of a domain is constructed from submodels. When only a small amount of data is available, the submodels consist of the data points themselves with similarity-based generalization.

As more data becomes available, submodels from more complex families may be constructed. The elementary induction step is to successively *merge* pairs of simple submodels to form more complex submodels. Because the new submodel is based on the combined data from the merged submodels, it may be reliably chosen from a more complex model space. This approach allows arbitrarily complex models to be constructed without overfitting. The resulting models adapt their representational power to the structure of the training data.

The search for submodels to merge is guided by an attempt to sacrifice as little of the sample likelihood as possible as a result of the merging process. This search can be done very efficiently if (a) a greedy search strategy can be used, and (b) likelihood computations can be done locally for each submodel and don't require global recomputation on each model update.

2.4.3 A curve-fitting example

This idea can be illustrated with task from geometrical data modeling. Consider the problem of modeling a curve in the plane by a combination of straight line segments. The likelihood in this case corresponds to the mean square error from each curve point to the nearest segment point.[8] A merging step in this case consists of replacing two segments by a single segment. We always choose that pair such that the merged segment increases the error the least. Figure 2.3 shows the approximations generated by this strategy. It does an excellent job at identifying the essentially linear portions of the curve and puts the boundaries between component models at the corners. While not shown in the figure, as repeated merges take place, more data is available for each segment. This would allow us to reliably fit submodels more complex than linear segments, such as Bezier curves. It is possible to reliably induce a representation which uses linear segments in some portions and higher order curves in others. Such models potentially have many parameters and would be subject to overfitting if they were learned directly rather than by going through merging steps.

Model merging has an obvious converse in iterative model *splitting*. In the curve example, this top-down approach would start with a single segment and repeatedly split it. This approach sometimes has to make decisions too early and often misses the corners in the curve. Although this is clearly domain-dependent, our experience has been that modeling approaches based on splitting tend to fit the structure of a domain less well than those based on merging. Model splitting approaches for grammatical models have been proposed by various authors and will be discussed in Chapter 3.

2.4.4 Knowing when to stop

A crucial factor in model merging is the stopping criterion. Since each merging step is a potential generalization step in the model space, the stopping criterion effectively determines how much generalization from the observed data takes place.

The criteria used can be of a theoretical, heuristic or practical nature. The following represents a non-exhaustive list of candidates:

[8] More precisely, the mean squared error is proportional to the log probability of the data under the assumption that each curve generates points normally distributed around them.

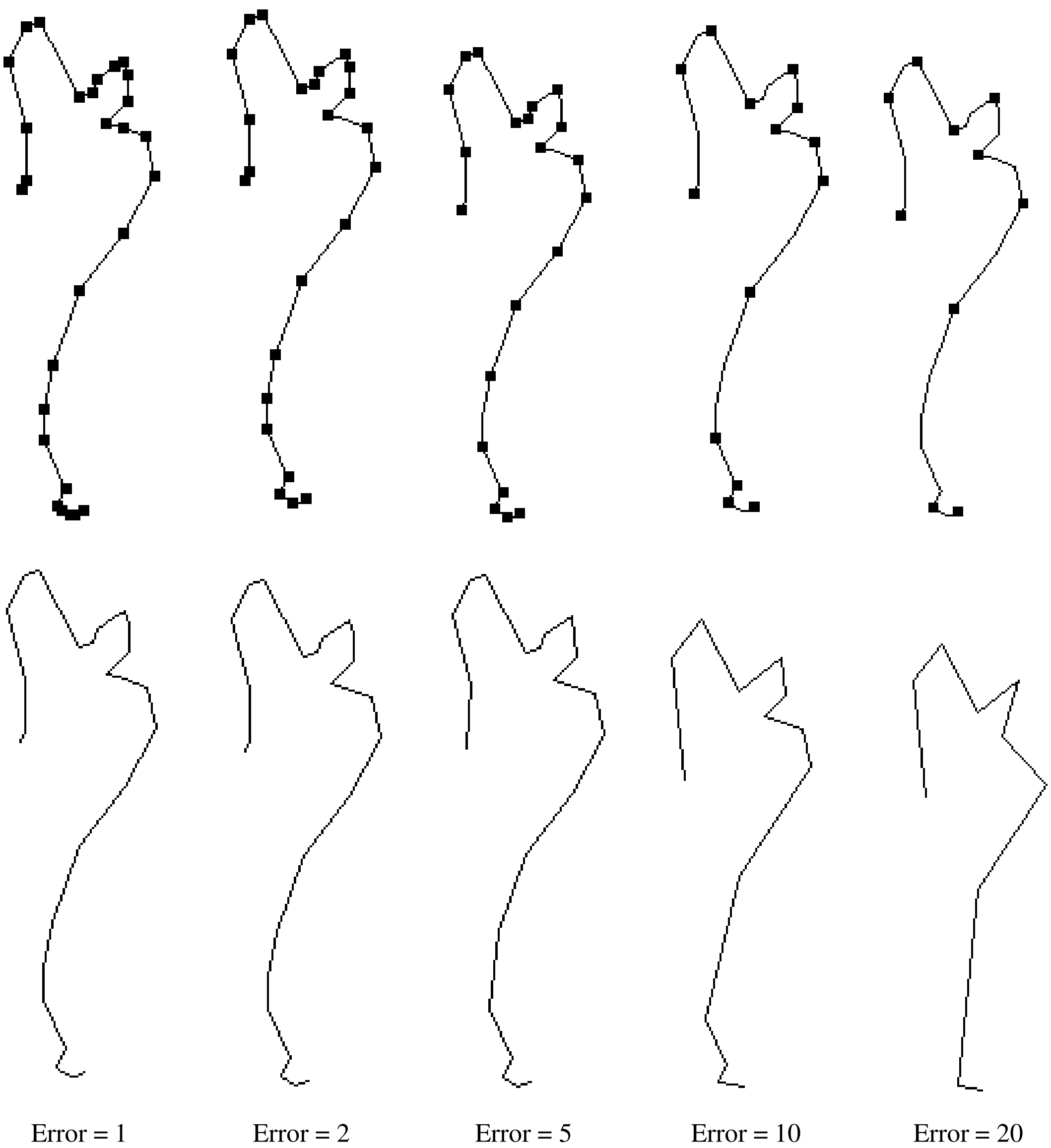


Figure 2.3: Approximation of a curve by best-first merging of segment models.
The top row shows the endpoints chosen by the algorithm at various levels of allowed error. The bottom row shows the corresponding approximation to the curve.

Global model characteristics If the target model has a known global characterization, i.e., in terms of its size, then merging can be stopped as soon as a model is reached that matches those criteria. Another reason to use global criteria such as size are resource limitations that the resulting model should fit.

Error tolerance The model's fit to the data can often be described in terms of loss measure (such as squared error or negative log likelihood). Merging then proceeds until a maximum error is exceeded.

Cross-validation Limiting the error on the training data is of course no guarantee for a similarly limited error on new data. Standard techniques to control the generalization error can be applied, such as cross-validation on a separate data set (i.e., merging stops when the model's error on the validation data increases).

Thresholding heuristics Instead of imposing fixed bounds on error or log likelihood one can alternatively use the *difference* in these measures that result from merging to control overgeneralization. It is empirically often the case that a merging step beyond the target model produces loss differentials that are large compared to those incurred in the merging steps leading up to the desired model.

The Bayesian approach to model merging developed in this thesis does not fit into any single one of these categories. Instead, it can be viewed as formalizing a *trade-off* between several of these criteria.

2.5 Bayesian Model Inference

2.5.1 The need for inductive bias

In the model merging framework, learning from sample data means generalizing from it. A simple maximum likelihood approach would not result in generalization, as the class of models is usually rich enough to allow a simple duplication of the sample data itself. This is different from traditional parameter estimation approaches where the fixed model structure constrains the maximum likelihood solution, thereby leading to generalization. The goal of model merging, of course, is to determine the right model structure from the data, instead of having it specified *a priori*.

An intuitive justification for prefering more general models with lower likelihood over ML models is that the former are *simpler* in some sense. In particular, since the ML models in model merging grow with the amount of data supplied, their size would be unbounded unless the sample space itself is finite.

Simplicity or complexity is an important and well-known form of *inductive bias* (Mitchell 1980). It is also known as 'Occam's Razor' following William of Occam's dictum not 'to multiply things beyond necessity,' i.e., that given two explanations for an empirical fact, the one with less assumptions is to be preferred. A crude form of simplicity metric is embodied in the model merging process itself: as submodels are merged, the total model *size*, suitably defined, typically decreases.

However, what is really needed is a principled approach to *trade off* the inductive bias towards simpler models against the fit of the model to the observed data. We have seen that it is not useful to fit the data perfectly; on the other hand, only maximizing simplicity would lead to models that bear no relation to the actual distribution generating the data. We therefore need a formal notion of both model complexity and fit to the data, such that the trade-off between the two can be quantified.

In this section we will present two such formalizations, Bayesian inference and inference by Minimum Description Length. These two approaches turn out to be essentially equivalent, but will provide two complementary conceptualizations of a particular form of probabilistic inductive bias.

The principles discussed here are by no means unique to the probabilistic scenario. For example, the futility of exactly matching the model to the data, and the resulting need for inductive bias, is demonstrated by Mitchell (1980) for the case of discrete classifier induction.

2.5.2 Posterior probabilities

We can express our *a priori* preferences regarding alternative models (for simplicity or otherwise) in probabilistic terms using the Bayesian notion of *prior probability*. We assume that there exists a distribution $P(M)$ independent of the data that assigns each model M an *a priori* (before the data) probability. This is clearly a probabilistic form of bias, as it presumably gives some models higher prior probability than others.

Given some data X we can then endeavor to find the model M that maximizes the *posterior probability* $P(M|X)$. Bayes' Law expresses the posterior as

$$P(M|X) = \frac{P(M)P(X|M)}{P(X)} \tag{2.13}$$

Since the data X is fixed, M_{MAP} maximizes $P(M)P(X|M)$, where $P(X|M)$ is the familiar likelihood. The resulting model also know as the MAP (maximum a posteriori probability) model.

This form of Bayesian model inference is therefore a generalization of the Maximum-Likelihood (ML) estimation method, as the prior $P(M)$ is combined with the usual likelihood term $P(X|M)$ for maximization purposes.

2.5.3 Bayesian Model Merging

It is also easy to modify any given likelihood-based model merging strategy to accommodate prior probabilities. Two basic changes need to be made:

- Each merging step should maximize the model posterior instead of the model likelihood.
- The stopping criterion is that no further increase in model posterior is possible through one choice of the possible merging steps.

As before, this Bayesian model merging variant carries out a greedy search of the model space, whose topology is implicitly defined by the merging operators. The goal of the search is now to find a local maximum of the posterior probability.

Note that given our search strategy there are really two forms of bias at work in this modified induction strategy:

- An explicit, *probabilistic bias* expressed by the prior $P(M)$.
- An implicit, heuristic bias as part of the choice of the topology of the search space (the *search bias*). Even if the merging operator(s) are defined such that all models in the space are reachable from the initial model, this bias is still significant due to the local and sequential nature of the posterior probability maximization.

We will return to the question of how to relax the search bias by using less constrained search methods (Section 3.4.5).

The remaining section of this chapter present some of the common mathematical tools needed in instantiating the Bayesian model merging approach in the domain of probabilistic grammars. The following chapters 3, 4 and 5 each describe one such instantiation.

2.5.4 Minimum Description Length

Bayesian inference based on posterior probabilities has an alternative formulation in terms off information-theoretic concepts. The dualism between the two formulations is useful both for a deeper understanding of the underlying principles, and for the construction of prior distributions (see Section 2.5.6 below).

The maximization of

$$P(M, X) = P(M)P(X|M)$$

implicit in Bayesian model inference is equivalent to minimizing

$$-\log P(M, X) = -\log P(M) - \log P(X|M) \quad .$$

Information theory tells us that the negative logarithm of the probability of a discrete event E is the optimal code word length for communicating an instance of E, so as to minimize the average code length of a representative message. Accordingly, the terms in the above equation can be interpreted as *message* or *description lengths*.

Specifically, $-\log P(M)$ is the description length of the model under the prior distribution; $-\log P(X|M)$ corresponds to a description of the data X using M as the model on which code lengths are based. The negative log of the joint probability can therefore be interpreted as the *total description length* of model and data.

Inference or estimation by *minimum description length (MDL)* (Rissanen 1983; Wallace & Freeman 1987) is thus equivalent to, and a useful alternative conceptualization of posterior probability maximization.[9]

[9] The picture become somewhat more complex when distributions over continuous spaces are involved. For those the corresponding MDL formulation has to consider the optimal granularity of the discrete encoding. We will can conveniently avoid this complication as the only formal use of description lengths will be to devise priors for *discrete* objects, namely, grammar structures.

2.5.5 Structure vs. parameter priors

A grammatical model is here described in two stages:

1. A model *structure* or *topology* is specified as a set of states, nonterminals, transitions, productions, etc. (depending on the type of model). These elements represent discrete choices as to which derivations from the grammar can have non-zero probability.

2. Conditional on a given structure, the model's continuous parameters are specified. These are typically multinomial parameters.

We will write $M = (M_S, \theta_M)$ to describe the decomposition of model M into the structure part M_S and the parameter part θ_M. The model prior $P(M)$ can therefore be written as

$$P(M) = P(M_S)P(\theta_M|M_S)$$

Even this framework leaves some room for choice: as discussed earlier, one may choose to make the structure specification very unconstrained, *e.g.*, by allowing all probability parameters to take on non-zero values, effectively pushing the structure specification into the parameter choice. Examples of this will be discussed in the following chapters.

2.5.5.1 Priors for multinomial parameters

Since the continuous parameters in the grammar types dealt with in this thesis are all from multinomial distributions, it is convenient to discuss some standard priors for this type of distribution at this point.

Each multinomial represents a discrete, finite probabilistic choice of some event. Let n be the number of choices in a multinomial and, $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$ the probability parameters associated with each choice (only $n-1$ of these parameters are free since $\sum_i \theta_i = 1$).

A standard prior for multinomials is the *Dirichlet distribution*

$$P(\boldsymbol{\theta}) = \frac{1}{B(\alpha_1, \ldots, \alpha_n)} \prod_{i=1}^{n} \theta_i^{\alpha_i - 1} \quad , \tag{2.14}$$

where $\alpha_1, \ldots, \alpha_n$ are parameters of the prior which can be given an intuitive interpretation (see below). The normalizing constant $B(\alpha_1, \ldots, \alpha_n)$ is the n-dimensional Beta function,

$$B(\alpha_1, \ldots, \alpha_n) = \frac{\Gamma(\alpha_1) \cdots \Gamma(\alpha_n)}{\Gamma(\alpha_1 + \cdots + \alpha_n)} \quad .$$

The *prior weights* α_i determine the bias embodied in the prior: the prior expectation of θ_i is $\frac{\alpha_i}{\alpha_0}$, where $\alpha_0 = \sum_i \alpha_i$ is the *total prior weight*.

One important reason for the use of the Dirichlet prior in the case of multinomial parameters (Cheeseman *et al.* 1988; Cooper & Herskovits 1992; Buntine 1992) is its mathematical expediency. It is a *conjugate prior*, *i.e.*, of the same functional form as the likelihood function for the multinomial. The likelihood for a sample from the multinomial with total observed outcomes $c_1, \ldots, c_n$ is given by equation (2.4). This means that the prior (2.14) and the likelihood (2.4) combine according to Bayes' law to give an expression for the posterior density that is again of the same form, namely:

$$P(\boldsymbol{\theta}|c_1, \ldots, c_n) = \frac{1}{B(c_1+\alpha_1, \ldots, c_n+\alpha_n)} \prod_{i=1}^{n} \theta_i^{c_i+\alpha_i-1} \quad . \tag{2.15}$$

Furthermore, it is convenient that the integral over the product of (2.14) and (2.4) has a closed-form solution.

$$\begin{aligned} \int_{\boldsymbol{\theta}} P(\boldsymbol{\theta}) P(c_1, \ldots, c_n|\boldsymbol{\theta}) d\boldsymbol{\theta} &= \frac{1}{B(\alpha_1, \ldots, \alpha_n)} \int_{\boldsymbol{\theta}} \prod_{i=1}^{n} \theta_i^{c_i+\alpha_i-1} d\boldsymbol{\theta} \\ &= \frac{B(c_1+\alpha_1, \ldots, c_n+\alpha_n)}{B(\alpha_1, \ldots, \alpha_n)} \end{aligned} \tag{2.16}$$

This integral will be used to compute the posterior for a given model structure, as detailed in Sections 2.5.7 and 3.4.3.

To get an intuition for the effect of the Dirichlet prior it is helpful to look at the two-dimensional case. For $n = 2$ there is only one free parameter, say $\theta_1 = p$, which we can identify with the probability of heads in a biased coin flip ($\theta_2 = 1 - p$ being the probability of tails). Assume there is no *a priori* reason to prefer either outcome, *i.e.*, the prior distribution should be symmetrical about the value $p = 0.5$. This symmetry entails a choice of α_i's which are equal, in our case $\alpha_1 = \alpha_2 = \alpha$. The resulting prior distribution is depicted in Figure 2.4(a), for various values of α. For $\alpha_i > 1$ the prior has the effect of adding $\alpha_i - 1$ 'virtual' samples to the likelihood expression, resulting in a MAP estimate of

$$\hat{\theta}_i = \frac{c_i + \alpha_i - 1}{\sum_j (c_j + \alpha_j - 1)} \quad . \tag{2.17}$$

For $0 < \alpha_i < 1$ the MAP estimate is biased towards the extremes of the parameter space, $\theta_i = 0$ and $\theta_i = 1$. For $\alpha_i = 1$ the prior is uniform and the MAP estimate is identical to the ML estimate.

Figure 2.4(b) shows the effect of varying amounts of data N (total number of samples) on the posterior distribution. With no data ($N = 0$) the posterior is identical to the prior, illustrated here for $\alpha = 2$. As N increases the posterior peaks around the ML parameter setting.

2.5.6 Description Length priors

The MDL framework is most useful for designing priors for the discrete, structural aspects of grammars. Any (prefix-free) coding scheme for models that assigns M a code length $\ell(M)$ can be used to induce a prior distribution over models with

$$P(M) \propto e^{-\ell(M)} \quad .$$

We can take advantage of this fact to design 'natural' priors for many domains. This idea will be extensively used with all grammar types.

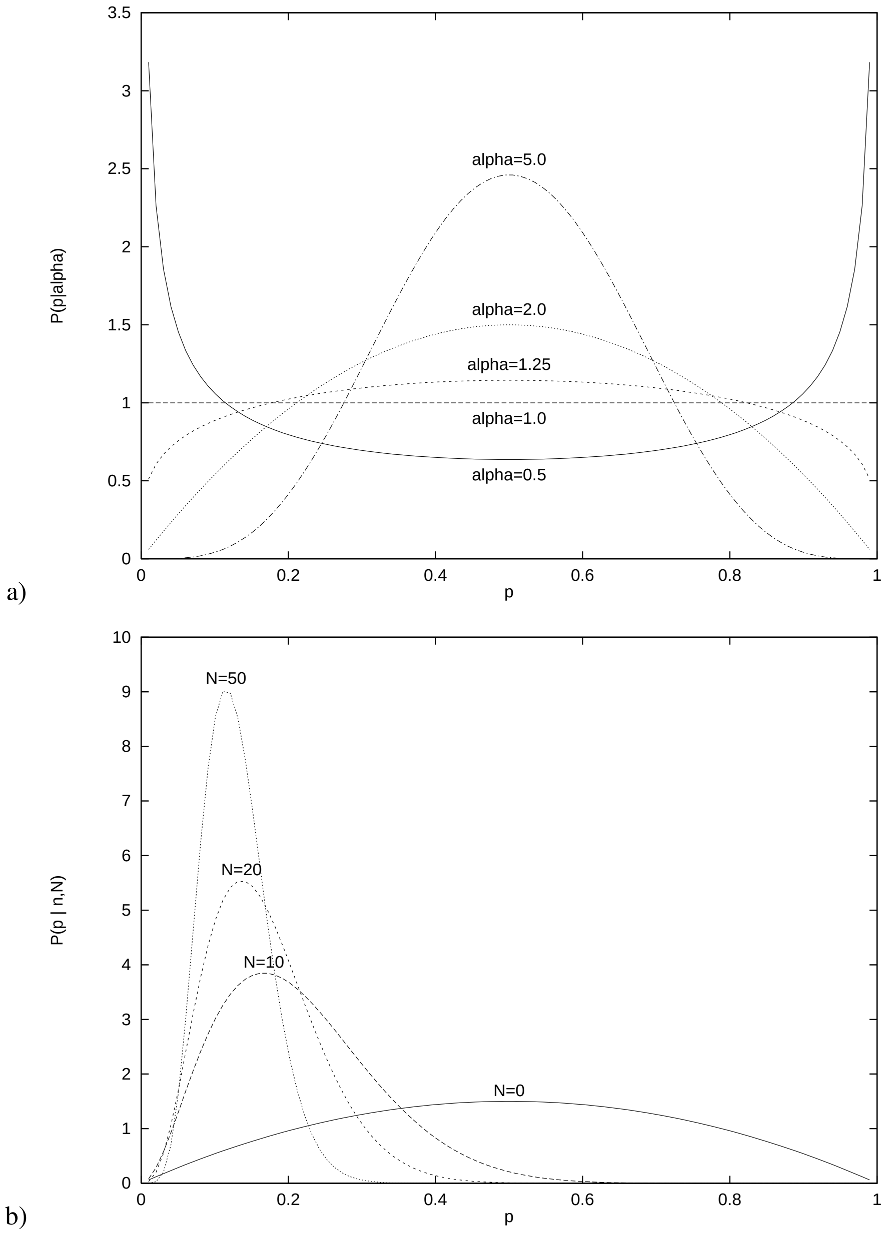


Figure 2.4: The two-dimensional symmetrical Dirichlet prior.

(a) Prior distributions for various prior weights α. (b) Posterior distributions for $\alpha = 2.0$ and various amounts of data $N = c_1 + c_2$ in the proportion $c_1/N = 0.1$.

2.5.7 Posteriors for grammar structures

The Bayesian approach in its simplest form computes the posterior probability of a fully specified model,

$$P(M|X) \propto P(M)P(X|M) \quad ,$$

and compares alternative models on that basis. If the goal is to find a single model that best represents the data, this approach amounts to a joint maximization of the posterior $P(M|X)$ over both the model structure M_S and its parameters θ_M.

Alternatively, we may want to infer a single grammar M and view it as a representative of a class of grammars obtained by varying the parameters θ_M according to their posterior distribution, $P(\theta_M|X, M_S)$. For example, when new data X' arrives, its probability is assessed as a weighted average over all parameter settings.

$$P(X'|X, M_S) = \int_{\theta_M} P(\theta_M|X, M_S)P(X'|M_S, \theta_M)d\theta_M \tag{2.18}$$

This is motivated by the Bayes-optimal solution to the transductive inference $P(X'|X)$, which would consist of summing not only over all possible parameter settings, but over all possible model structures as well:

$$P(X'|X) = \sum_{M_S} P(M_S|X) \int_{\theta_M} P(\theta_M|X, M_S)P(X'|M_S, \theta_M)d\theta_M \tag{2.19}$$

Choosing a single structure is an approximation to the full Bayesian solution, *i.e.*, averaging only over a part of the full model space. To optimize this approximation we should choose the model structure M_S which maximizes the associated posterior weight in equation (2.19),

$$P(M_S|X) \int_{\theta_M} P(\theta_M|X, M_S)d\theta_M = P(M_S|X) \quad .$$

This reasoning suggests changing the objective from maximizing the joint posterior probability of the structure and parameters together, to maximizing the posterior probability of the model structure alone. The desired quantity is obtained by integrating out the 'nuisance' variable θ_M:

$$\begin{aligned} P(M_S|X) &= \frac{P(M_S)P(X|M_S)}{P(X)} \\ &= \frac{P(M_S)}{P(X)} \int_{\theta_M} P(X, \theta_M|M_S)d\theta_M \\ &= \frac{P(M_S)}{P(X)} \int_{\theta_M} P(\theta_M|M_S)P(X|M_S, \theta_M)d\theta_M \quad . \end{aligned} \tag{2.20}$$

Unfortunately, there is usually no way to compute this integral exactly in closed form, since $P(X|M_S, \theta_M)$ is a sum over all possible derivations by M that can generate X, and whose respective probabilities vary with θ_M. In practice we resort to Viterbi approximations of the quantities involved, as described specifically for each model type in the following chapters.

Chapter 3

Hidden Markov Models

3.1 Introduction and Overview

Hidden Markov Models (HMMs) are a popular method for modeling stochastic sequences with an underlying finite-state structure. Some of their first uses were in the area of cryptanalysis and they are now the model of choice for speech recognition (Rabiner & Juang 1986). Recent applications include part-of-speech tagging (Kupiec 1992b) and protein classification and alignment (Haussler *et al.* 1992; Baldi *et al.* 1993). Because HMMs can be seen as probabilistic generalizations of non-deterministic finite-state automata they are also of interest from the point of view of formal language induction.

For most modeling applications it is not feasible to specify HMMs by hand. Instead, the HMM needs to be at least partly estimated from available sample data. All of the applications mentioned crucially involve *learning*, or adjusting the HMM to such data. Standard HMM estimation techniques assume knowledge of the model size and structure (or topology) and proceed to optimize the continuous model parameters using well-known statistical techniques. Section 3.2 defines the HMM formalism and gives an overview of these standard estimation methods.

In contrast to traditional HMM estimation based on the Baum-Welch technique (Baum *et al.* 1970), our model merging method adjusts the model topology to the data. The merging operator for HMMs is very simple: it is realized by collapsing two model states (as well as their transitions and emissions). The resulting algorithm is described in Section 3.3.

As a result of our implementation and applications of the merging algorithm, a number of crucial efficiency improvements, approximations and heuristics have been developed. These are discussed in Section 3.4.

Model merging for HMMs is related to ideas that have appeared in the literature, in some cases for considerable time. Section 3.5 discusses some of these links to related work and compares the various approaches.

The Bayesian HMM merging algorithm is evaluated experimentally using both artificial and realistic applications in Section 3.6. We compare its structure-induction capabilities to those of the Baum-Welch method, and find that it produces models that have better generalization and/or are more compact. In particular, applications in the area of phonetic word modeling for speech recognition show that HMM merging can be an effective and efficient tool in practice.

Section 3.7 summarizes the chapter and points to point to continuations of this line of research.

3.2 Hidden Markov Models

3.2.1 Definitions

We now define formally the HMMs introduced in Section 2.3.4.[1]

Definition 3.1 A *(discrete output, first-order) Hidden Markov Model* is specified by

a) a set of *states* $\mathcal{Q}$,

b) an *output alphabet* Σ,

c) an *initial state* q_I,

d) a *final state* q_F,

and a set of probability parameters.

e) *Transition probabilities* $p(q \rightarrow q')$ specify the probability that state q' follows q, for all $q, q' \in \mathcal{Q}$.

f) *Emission (output) probabilities* $p(q \uparrow \sigma)$ specify the probability that symbol σ is emitted while in state q, for all $q \in \mathcal{Q}$ and $\sigma \in \Sigma$.

By the *structure* or *topology* of an HMM we mean its states $\mathcal{Q}$, its outputs Σ, a subset of its transitions $q \rightarrow q'$ with $p(q \rightarrow q') = 0$ and a subset of its emissions $q \uparrow \sigma$ with $p(q \uparrow \sigma) = 0$. In other words, an HMM topology specifies a subset of the potential transitions and emissions which are guaranteed to have zero probability, and leaves the remaining probabilities unspecified.

We use superscripts on states q^t and emissions σ^t to denote discrete time indices in the generation of an output sequence. Therefore,

$$p(q \rightarrow q') = p((q')^{t+1} | q^t), \quad t = 0, 1, 2, \ldots$$

and

$$p(q \uparrow \sigma) = p(\sigma^t | q^t), \quad t = 0, 1, 2, \ldots$$

The initial state q_I occurs at the beginning of any state sequence and the final state q_F at the end of any complete state sequence. Neither q_I nor q_F can occur anywhere else, and they do not emit symbols. For convenience we assume $q_I, q_F \notin \mathcal{Q}$.

[1]Where possible we try to keep the notation consistent with Bourlard & Morgan (1993).

Definition 3.2 An HMM M is said to *generate* a string $x = x_1x_2 \ldots x_\ell \in \Sigma^*$ if and only if there is a state sequence, or *path*, $q_1q_2 \ldots q_\ell \in \mathcal{Q}^*$ with non-zero probability, such that q_t outputs x_t with non-zero probability, for $t = 1, \ldots, \ell$. The *probability of a path* (relative to x) is the product of all transition and emission probabilities along it.

The *probability* $P(x|M)$ of a string x given an HMM M is computed as the sum of the probabilities of all paths that generate x:

$$P(x|M) = \sum_{q_1 \ldots q_\ell \in \mathcal{Q}^\ell} p(q_I \to q_1)p(q_1 \uparrow x_1)p(q_1 \to q_2) \ldots p(q_\ell \uparrow x_\ell)p(q_\ell \to q_F) \tag{3.1}$$

There are some conditions on the probability parameters in an HMM necessary to ensure that $P(x|M)$ as defined above forms a proper distribution over the set of finite strings.

- Transition probabilities on each state have to sum to unity:
$$\sum_{q' \in \mathcal{Q}} p(q \to q') = 1$$
- Emission probabilities on each state have to sum to unity:
$$\sum_{\sigma \in \Sigma} p(q \uparrow \sigma) = 1$$
- All states reachable from q_I by a path with non-zero probability must also have a path to q_F of non-zero probability. (Otherwise there would be dead-end states that 'capture' some of the total probability mass without contributing to the total distribution $P(x|M)$.)

These well-formed Ness conditions are always satisfied when obtaining HMMs through one of the standard estimation algorithms or through model merging, so they are of no immediate concern to us.

3.2.2 HMM estimation

The Baum-Welch estimation method for HMMs (Baum *et al.* 1970) assumes a certain topology and adjusts the parameters so as to maximize the model likelihood on the given samples. If the structure is only minimally specified (*i.e.*, all probabilities can assume non-zero values) then this method can potentially *find* HMM structures by setting a subset of the parameters to zero (or close enough to zero so that pruning is justified).

The fundamental problem in HMM estimation is that the state variables are not directly observable. If they were, *i.e.*, if we could observe sequences of states $q_1q_2 \ldots q_\ell$ in addition to the outputs, estimation of the probability parameters would be straightforward. One could collect sufficient statistics,

$$\begin{aligned} c(q \to q') &= \text{number of transitions from state } q \text{ to } q'\text{, for all } q, q' \in \mathcal{Q} \\ c(q \uparrow \sigma) &= \text{number of outputs of } \sigma \text{ from state } q\text{, for all } q \in \mathcal{Q}, \sigma \in \Sigma, \end{aligned}$$

and set the model parameters to their maximum likelihood values:

$$\hat{p}(q \to q') = \frac{c(q \to q')}{\sum_{s \in \mathcal{Q}} c(q \to s)} \tag{3.2}$$

$$\hat{p}(q \uparrow \sigma) = \frac{c(q \uparrow \sigma)}{\sum_{\rho \in \Sigma} c(q \uparrow \rho)}. \tag{3.3}$$

The missing state observations are just a special case hidden variables, as discussed in Section 2.3.2. Consequently, the EM method can be instantiated in this case: we replace the unknown transition and output frequencies by their expected values given a current model estimate and the sample output sequences. For each sample sequence x we compute the posterior probability $P(q^t|x, M)$ that the path generating x passes through state q at time t. This can be done by an efficient $O(\ell|\mathcal{Q}|^2)$ dynamic programming algorithm know as the *forward-backward* algorithm (see, e.g., (Rabiner & Juang 1986)). From $P(q^t|x, M)$ for all q and t, it is then straightforward to compute the posterior expectations

$$\begin{aligned} \hat{c}(q \to q') &= E[c(q \to q')|X, M] \\ \hat{c}(q \uparrow \sigma) &= E[c(q \uparrow \sigma)|X, M]. \end{aligned}$$

The model parameters are then maximized with respect to the expectations $\hat{c}$ instead of the unknown values c. Re-estimating parameters affects the expectations, so $\hat{c}$ has to be recomputed, parameters estimated again, *etc.*, until a fixed point is reached.

As with EM in the general, the Baum-Welch method is not fool-proof: since it uses what amounts to a hill-climbing procedure that is only guaranteed to find a local likelihood maximum, the result of Baum-Welch estimation may turn out to be sub-optimal. In particular, results will depend on the initial values chosen for the model parameters. Several examples of this phenomenon will be seen in Section 3.6.1.

3.2.3 Viterbi approximation

A frequently used approximation in HMM estimation is to proceed as if each sample comes from only a single path through the model, *i.e.*, all paths except the most likely one are assumed to have zero or negligible probability. The most likely, or *Viterbi path* (after Viterbi (1967)) is the one that maximizes the summand in equation (3.1):

$$V(x|M) = \operatorname*{argmax}_{q_1 \ldots q_\ell \in \mathcal{Q}^\ell} p(q_I \to q_1) p(q_1 \uparrow x_1) p(q_1 \to q_2) \ldots p(q_\ell \uparrow x_\ell) p(q_\ell \to q_F) \tag{3.4}$$

Let $V_i(x|M)$ denote the ith state in $V(x|M)$, with $V_0(x|M) = q_I$ and $V_{\ell+1}(x|M) = q_F$ for convenience.

By neglecting all paths except $V(x|M)$, the statistics used in re-estimating transitions and emissions become sums of 0's and 1's. The resulting approximated estimates are

$$\begin{aligned} \hat{c}(q \to q') &= \sum_{x \in X} \sum_{i=0}^{\ell} \delta(q, V_i(x|M)) \delta(q', V_{i+1}(x|M)) \\ \hat{c}(q \uparrow \sigma) &= \sum_{x \in X} \sum_{i=1}^{\ell} \delta(q, V_i(x|M)) \delta(\sigma, x_i) \quad , \end{aligned}$$

where the Kronecker delta $\delta(x, y)$ is 1 if $x = y$ and 0 otherwise.

We mention the Viterbi approximation here because it also turns out to be very useful in an efficient implementation of the HMM induction algorithm described in later sections.

3.3 HMM Merging

We can now describe the application of model merging to HMMs. First, the merging process is described and illustrated using the simple, likelihood-only merging strategy. We then discuss priors for HMMs, and finally give a modified algorithm using the resulting posterior probabilities. Implementation details are filled in in Section 3.4.

3.3.1 Likelihood-based HMM merging

The model merging method requires four major elements:

1. A method to construct an initial model from data.
2. A way to merge submodels.
3. An error measure to compare the goodness of various candidates for merging and to limit the generalization.
4. A strategy to pick merging operators, i.e., search the model space.

These elements can be translated to the HMM domain as follows:

1. An initial HMM is constructed as a disjunction of all observed samples. Each sample is represented by dedicated HMM states such that the entire model generates all and only the observed strings.
2. The merging step combines individual HMM states and gives the combined state emission and transition probabilities which are weighted averages of the corresponding distributions for the states which have been merged.
3. The simplest error is the negative logarithm of the model likelihood. Later we show how this is generalized to a Bayesian posterior model probability criterion that provides a principled basis for limiting generalization.
4. The default search strategy is greedy or best-first search, i.e., at each step the states with give the best score according to the evaluation function is chosen (other strategies are discussed in Section 3.4.5). There is also an issue as to how the incorporation of samples is scheduled relative to the merging itself. For the time being we assume that all samples are incorporated before merging starts, but a more practical alternative will be given below in Section 3.3.5.

To obtain an initial model from the data, we first construct an HMM which produces exactly the input strings. The start state has as many outgoing transitions as there are strings and each string is represented by a unique path with one state per sample symbol. The probability of entering these paths from the start state is uniformly distributed. Within each path there is a unique transition to the next state, with probability 1. The emission probabilities are 1 for each state to produce the corresponding symbol.

The initial model resulting from this procedure has the property that it assigns each sample a probability equal to its relative frequency, and is therefore a maximum likelihood model for the data, as is generally true for initial models in the model merging methodology. In this sense the initial HMM is also the most *specific* model compatible with the data (modulo weak equivalence among HMMs).

The merging operation, repeatedly applied to pairs of HMM states, preserves the ability to generate all the samples accounted for by the initial model. However, new, unobserved strings may also be generated by the merged HMM. This in turn means that the probability mass is distributed among a greater number (possibly an infinity) of strings, as opposed to just among the sample strings. The algorithm therefore *generalizes* the sample data.

The drop in the likelihood relative to the training samples is a measure of how much generalization occurs. By trying to minimize the change in likelihood, the algorithm performs repeated *conservative* generalizations, until a certain threshold is reached. We will see later that the trade-off between model likelihood and generalization can be recast in Bayesian terms, replacing the simple likelihood thresholding scheme by the maximization of posterior model probability.

3.3.2 An example

Consider the regular language $(ab)^+$ and two samples drawn from it, the strings ab and $abab$. Using the above procedure, the algorithm constructs the initial model M_0 depicted in Figure 3.1.

From this starting point, we can perform two merging steps without incurring a drop in the model likelihood.[2] First, states 1 and 3 are merged (M_1), followed by 2 and 4 (M_2).

Merging two states entails the following changes to a model:

- The old states are removed and the new 'merged' state is added. The old states are called the 'parent' states.
- The transitions from and to the old states are redirected to the new state. The transition probabilities are adjusted to maximize the likelihood.
- The new state is assigned the union of the emissions of the old states and the emission probabilities are adjusted to maximize the likelihood.

In this example we use the convention of numbering the merged state with the smaller of the indices of its parents.

[2] Actually there are two symmetrical sequences of merges with identical result. We have arbitrarily chosen one of them.

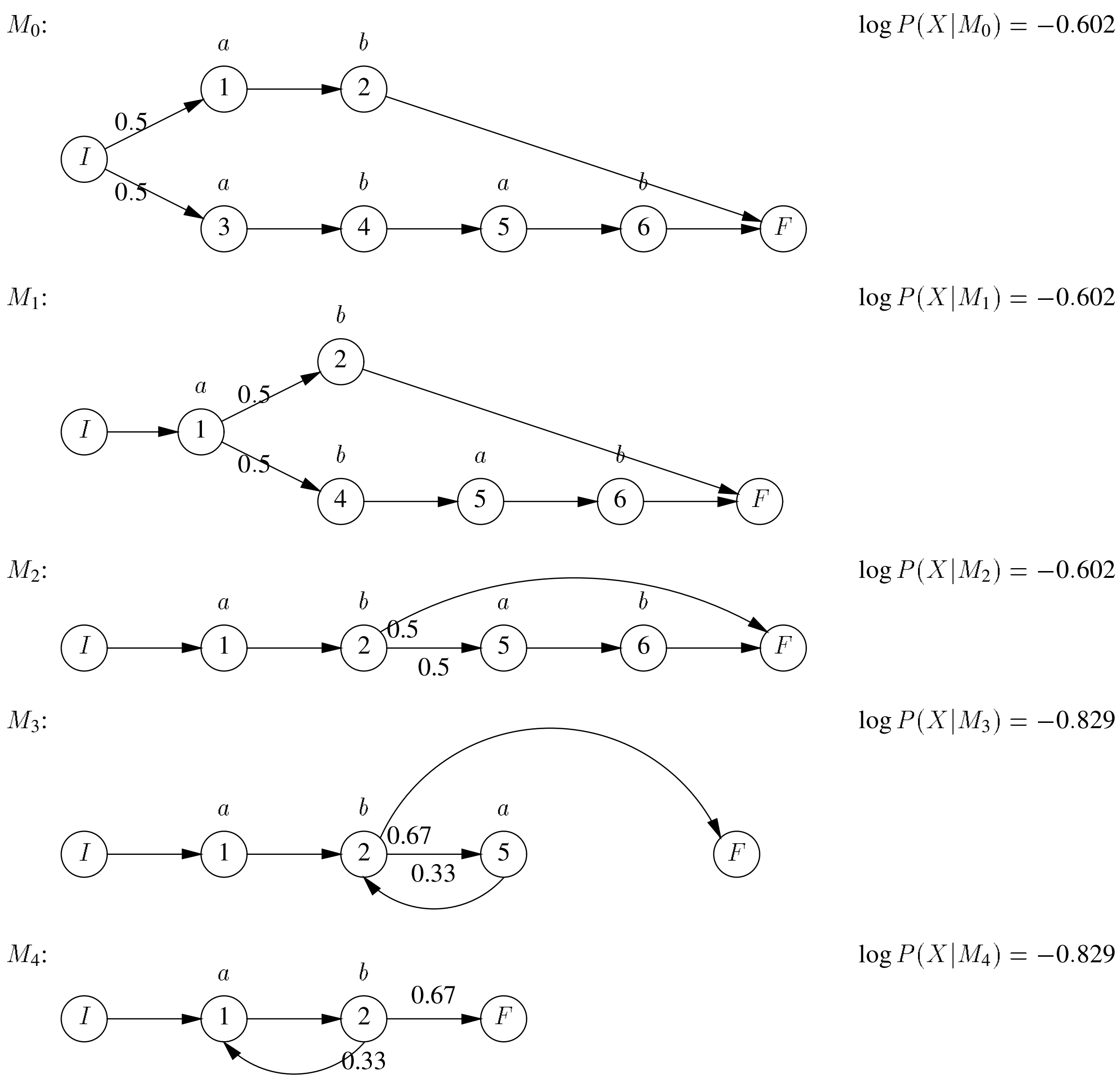


Figure 3.1: Sequence of models obtained by merging samples $\{ab, abab\}$.

All transitions without special annotations have probability 1. Output symbols appear above their respective states and also carry an implicit probability of 1. For each model, the log likelihood (base 10) is given.

Returning to the example, we now chose to merge states 2 and 6 (M_3). This step decreases the log likelihood (from -0.602 to -0.829) but it is the smallest decrease that can be achieved by any of the potential merges.

Following that, states 1 and 5 can be merged without penalty (M_4). The resulting HMM is the minimal model generating the target language $(ab)^+$, but what prevents us from merging further, to obtain an HMM for $\{ab\}^+$?

It turns out that merging the remaining two states reduces the likelihood much more drastically than the previous, 'good' generalization step, from -0.829 to -3.465 (*i.e.*, three decimal orders of magnitude). A preliminary answer, therefore, is to set the threshold small enough to allow only desirable generalizations. A more satisfactory answer is provided by the Bayesian methods described below.

Note that further data may well justify the generalization to a model for $\{ab\}^+$. This *data-driven* character is one of the central aspects of model merging.

A domain-specific justification for model merging in the case of HMMs applies. It can be seen from the example that the structure of the generating HMM can always be recovered by an appropriate sequence of state merges from the initial model, provided that the available data 'covers' all of the generating model, *i.e.*, each emission and transition is exercised at least once. Informally, this is because the initial model is obtained by 'unrolling' the paths used in generating the samples in the target model. The iterative merging process, then, is an attempt to undo the unrolling, tracing a search through the model space back to the generating model. Of course, the best-first heuristic is not guaranteed to find the appropriate sequence of merges, or, less critically, it may result in a model that is only weakly equivalent to the generating model.

3.3.3 Priors for Hidden Markov Models

From the previous discussion it is clear that the choice of the prior distribution is important since it is the term in (2.13) that drives generalization. We take the approach that priors should be subject to experimentation and empirical comparison of their ability to lead to useful generalization. The choice of a prior represents an intermediate level of probabilistic modeling, between the global choice of model formalism (HMMs, in our case) and the choice of a particular instance from a model class (*e.g.*, a specific HMM structure and parameters). The model merging approach ideally replaces the usually poorly constrained choice of low-level parameters with a more robust choice of (few) prior parameters. As long as it doesn't assign zero probability to the correct model, the choice of prior is eventually overwhelmed by a sufficient amount of data. In practice, the ability to *find* the correct model may be limited by the search strategy used, in our case, the merging process.

HMMs are a special kind of parameterized graph structure. Unsurprisingly, many aspects of the priors discussed in this section can be found in Bayesian approaches to the induction of graph-based models in other domains (*e.g.*, Bayesian networks (Cooper & Herskovits 1992; Buntine 1991) and decision trees (Buntine 1992)).

3.3.3.1 Structural vs. parameter priors

As discussed in Section 2.5.5, an HMM may be specified as a combination of structure and continuous parameters. For HMMs the structure or topology is given by as a set of states, transitions and emissions. Transitions and emissions represent discrete choices as to which paths and outputs can have non-zero probability in the HMM.

Our approach is to compose a prior for both the structure and the parameters of the HMM as a product of independent priors for each transition and emission multinomial, possibly along with a global factor. Although the implicit independence assumption about the parameters of different states is clearly a simplification, it shouldn't introduce any systematic bias toward any particular model structure. It does, however, greatly simplify the computation and updating of the global posteriors for various model variants, as detailed in Section 3.4.

The global prior for a model M thus becomes a product

$$P(M) = P(M_G) \prod_{q \in \mathcal{Q}} P(M_S^{(q)}|M_G) P(\theta_M^{(q)}|M_G, M_S^{(q)}) \tag{3.5}$$

where $P(M_G)$ is a prior for global aspects of the model structure (including, *e.g.*, the number of states), $P(M_S^{(q)})$ is a prior contribution for the structure associated with state q, and $P(\theta_M^{(q)}|M_S^{(q)})$ is a prior on the parameters (transition and emission probabilities) associated with state q.

Unless otherwise noted, the global factor $P(M_G)$ is assumed to be unbiased, and therefore ignored in the maximization.

3.3.3.2 Parameter priors for HMMs

Since HMM transitions and emission probabilities are conceptually multinomials, one for each state, we apply the Dirichlet prior discussed in Section 2.5.5.1. What the parameters are exactly depends on the structure-vs.-parameter trade-off.

Narrow parameter priors A natural application of the Dirichlet prior is as a prior distribution over each set of multinomial parameters within a given HMM structure M_S. Relative to equation (3.5), the parameters of a state q with $n_t^{(q)}$ transitions and $n_e^{(q)}$ emissions contribute a factor

$$P(\theta_M^{(q)}|M_G, M_S^{(q)}) = \frac{1}{B(\alpha_t, \ldots, \alpha_t)} \prod_{i=1}^{n_t^{(q)}} \theta_{qi}^{\alpha_t - 1} \frac{1}{B(\alpha_e, \ldots, \alpha_e)} \prod_{j=1}^{n_e^{(q)}} \theta_{qj}^{\alpha_e - 1} \quad . \tag{3.6}$$

Here θ_{qi} are the transition probabilities at state q, i ranging over the states that can follow q; θ_{qj} are the emission probabilities in state q, j ranging over the outputs emitted by q. α_t and α_e are the prior weights for transitions and emissions, respectively, and can be chosen to introduce more or less bias towards a uniform assignment of the parameters.

Broad parameter priors In the preceding version the parameters were constrained by the choice of a model structure M_S. As indicated earlier, one may instead let the parameters range over all potential transitions (all states in the model) and emissions (all elements of the output alphabet). Dirichlet priors as in equation (3.6) can still be used, using $n_t^{(q)} = |\mathcal{Q}|$ and $n_e^{(q)} = |\Sigma|$ for all states q.

One interesting aspect of this approach is that at least the emission prior weights can be chosen to be non-symmetrical, with prior means

$$E[\theta_i] = \frac{\alpha_i}{\sum_j \alpha_j}$$

adjusted so as to match the empirical fraction of symbol occurrences in the data. This 'empirical Bayes' approach is similar to the setting of prior class probability means in Buntine (1992).

We are already working on the assumption that transitions and emissions are *a priori* independent of each other. It is therefore in principle possible to use any combination of broad and narrow parameter priors, although a full exploration of the possibilities remains to be done.[3]

3.3.3.3 Structure priors for HMMs

In the case of broad parameter priors the choice of transitions and emissions is already subsumed by the choice of parameters. The only structural component left open in this case is the number of states $|\mathcal{Q}|$. For example, one might add an explicit bias towards a smaller number of states by setting

$$P(M_S) \propto C^{-|\mathcal{Q}|}$$

for some constant $C > 1$. However, as we will see below, the state-based priors by themselves produce a tendency towards reducing the number of states as a result of Bayesian 'Occam factors' (Gull 1988).

In the case of narrow parameter priors we need to specify how the prior probability mass is distributed among all possible model topologies with a given number of states. For practical reasons it is desirable to have a specification that can be described as a product of individual state-based distributions. This leads to the following approach.

As for the transitions, we assume that each state has on average a certain number of outgoing transitions, n_t. We don't have a reason to prefer any of the $|\mathcal{Q}|$ possible target states *a priori*, so each potential transition will be assessed a prior probability of existence of $p_t = \frac{n_t}{|\mathcal{Q}|}$. Similarly, each possible emission will have a prior probability of $p_e = \frac{n_e}{|\Sigma|}$, where n_e is the prior expected number of emissions per state.

The resulting structural contribution to the prior for a state q becomes

$$P(M_S^{(q)}|M_G) = p_t^{n_t^{(q)}} (1-p_t)^{|\mathcal{Q}|-n_t^{(q)}} p_e^{n_e^{(q)}} (1-p_t)^{|\Sigma|-n_e^{(q)}} \quad . \tag{3.7}$$

As before, $n_t^{(q)}$ represents the number of transitions from state q, and $n_e^{(q)}$ the number of its emissions.

In MDL terms, the structural prior (3.7) corresponds to a HMM coding scheme in which each transition is encoded by $-\log p_t$ bits, and each emission with $-\log p_e$ bits. Potential transitions and emissions that are missing each take up $-\log(1-p_t)$ and $-\log(1-p_e)$ respectively.

[3] In the experiments reported later only narrow parameters priors, combined with simple MDL structure priors are used. The details can be found in the relevant sections.

Description Length priors We can use the MDL framework as discussed in Section 2.5.6 to derive simple priors for HMM structures from various coding schemes. For example, a natural way to encode the transitions and emissions in an HMM is to simply enumerate them. Each transition can be encoded using $\log(|\mathcal{Q}|+1)$ bits, since there are $|\mathcal{Q}|$ possible transitions, plus a special 'end' marker which allows us not to encode the missing transitions explicitly. The total description length for all transitions from state q is thus $n_t^{(q)} \log(|\mathcal{Q}| + 1)$. Similarly, all emissions from q can be coded using $n_e^{(q)} \log(|\Sigma| + 1)$ bits.[4]

The resulting prior

$$P(M_S^{(q)}|M_G) \propto (|\mathcal{Q}| + 1)^{-n_t^{(q)}}(|\Sigma| + 1)^{-n_e^{(q)}} \tag{3.8}$$

has the property that small differences in the number of states matter little compared to differences in the total number of transitions and emissions.

We have seen in Section 2.5.7 that the preferred criterion for maximization is the posterior of structure $P(M_S|X)$, which requires integrating out the parameters θ_M. In Section 3.4 we give a solution for this computation that relies on the approximation of sample likelihoods by Viterbi paths.

3.3.4 Why are smaller HMMs preferred?

Intuitively, we want an HMM induction algorithm to prefer 'smaller' models over 'larger' ones, other things being equal. This can be interpreted as a special case of 'Occam's razor,' or the scientific maxim that simpler explanations are to be preferred unless more complex explanations are required to explain the data.

Once the notions of model size (or explanation complexity) and goodness of explanation are quantified, this principle can be modified to include a trade-off between the criteria of simplicity and data fit. This is precisely what the Bayesian approach does, since in optimizing the product $P(M)P(X|M)$ a compromise between simplicity (embodied in the prior) and fit to the data (high model likelihood) is found.

But how is it that the HMM priors discussed in the previous section lead to a preference for 'smaller' or 'simpler' models? Two answers present themselves: one has to do with the general phenomenon of 'Occam factors' found in Bayesian inference; the other is related, but specific to the way HMMs partition data for purposes of 'explaining' it. We will discuss each in turn.

3.3.4.1 Occam factors

Consider the following scenario. Two pundits, M_1 and M_2, are asked for their predictions regarding an upcoming election involving a number of candidates. Each pundit has his/her own 'model' of the political process. We will identify these models with their respective proponents, and try to evaluate each according

[4] The basic idea of encoding transitions and emissions by enumeration has various more sophisticated variants. For example, one could base the enumeration of transitions on a canonical ordering of states, such that only $\log(n+1) + \log n + \cdots + log(n - n_t + 1)$ bits are required. Or one could use the k-out-of-n-bit integer coding scheme described in Cover & Thomas (1991) and used for MDL inference in Quinlan & Rivest (1989). Any reasonable Bayesian inference procedure should not be sensitive to such minor difference in the prior, unless it is used with too little data. Our goal here is simply to suggest priors that have reasonable qualitative properties, and are at the same time computationally convenient.

to Bayesian principles. M_1 predicts that only three candidates, A, B, and C have a chance to win, each with probability $\theta_{A_1} = \theta_{B_1} = \theta_{C_1} = \frac{1}{3}$. M_2 on the hand gives only A and B a realistic chance, with probability $\theta_{A_2} = \theta_{B_2} = \frac{1}{2}$. Candidate B turns out to be the winner. What is the posterior credibility of each pundit?

We marginalize over the (discrete) parameter space of each pundit's predictions. The 'data' X is the outcome of B's winning.

$$
\begin{aligned}
P(M_1|X) &\propto P(M_1)[P(A|M_1)P(X|A) + P(B|M_1)P(X|B) + P(C|M_1)P(X|C)] \\
&= P(M_1)[\theta_{A_1} \cdot 0 + \theta_{B_1} \cdot 1 + \theta_{C_1} \cdot 0] \\
&= P(M_1)\frac{1}{3} \\
P(M_2|X) &\propto P(M_2)[P(A|M_2)P(X|A) + P(B|M_2)P(X|B)] \\
&= P(M_2)[\theta_{A_2} \cdot 0 + \theta_{B_2} \cdot 1] \\
&= P(M_2)\frac{1}{2}
\end{aligned}
$$

Assuming that there is no *a priori* preference, $P(M_1) = P(M_2)$, we conclude that M_2 is more likely *a posteriori*. This result, of course, just confirms our intuition that a prophet whose predictions are specific (and true) is more credible than one whose predictions are more general.

The ratio between the allowable range of a model's parameters *a posterior* and *a priori* is known as the *Occam factor* (Gull 1988). In the discrete case these ranges are just the respective numbers of possible parameter settings: $\frac{1}{3}$ versus $\frac{1}{2}$ in the example. For continuous model parameters, the Occam factor penalizes those models in which the parameters have a larger range, or where the parameter space has a higher dimensionality. (This is how the Bayesian approach avoids always picking the model with the largest number of free parameters, which leads to overfitting the data.)

3.3.4.2 Effective amount of data per state

Prior to implementing (an approximation to) the full computation of the structure posterior for HMMs as dictated by equation (2.20), we had been experimenting with a rather crude heuristic that simply compared the likelihoods for alternative models, but evaluated *at the MAP parameter settings*. The prior used was a Dirichlet of the broad type discussed in Section 3.3.3.2. As a result, the structural prior itself, which favors smaller configurations, was completely missing. Surprisingly at first, this alone produced a preference for smaller models.

The intuitive reason for this is a combination of two phenomena, one of which is particular to HMMs. As is true in general, the MAP point migrates towards the maximum likelihood setting as the amount of data increases (Figure 2.4(b)). But in the case of HMMs, the effective amount of data *per state* increases as states are merged! In other words, as the number of states in an HMM shrinks, but total amount of data remains constant, each state will get to 'see' more of the data, on average. Therefore, merging the right states will cause some states to have more data available to them, allowing the likelihood to come closer to its maximum value.[5]

[5] Similar principles apply in other model merging applications in which the model effectively partitions the data for the purpose of

3.3.5 The algorithm

After choosing a set of priors and prior parameters, it is conceptually straightforward to modify the simple likelihood-based algorithm presented in Section 3.3.1 to accommodate the Bayesian approach. The best-first HMM merging algorithm takes on the following generic form.

Best-first merging (batch version)

A. Build the initial, maximum-likelihood model M_0 from the dataset X.

B. Let $i := 0$. Loop:

1. Compute a set of candidate merges K among the states of model M_i.
2. For each candidate $k \in K$ compute the merged model $k(M_i)$, and its posterior probability $P(k(M_i)|X)$.
3. Let k^* be the merge that maximizes $P(k(M_i)|X)$. Then let $M_{i+1} := k^*(M_i)$.
4. If $P(M_{i+1}|X) < P(M_i|X)$, return M_i as the induced model.
5. Let $i := i + 1$.

In this formulation 'model' can stand for either 'model structure + parameters' or, as suggested in Section 2.5.7, just 'model structure.' In discussing our implementation and results we assume the latter unless explicitly stated otherwise.

Note that many of the computational details are not fleshed out here. Important implementation strategies are described in Section 3.4.

The number of potential merges in step B.2, $|K|$, is the biggest factor in the total amount of computation performed by the algorithm. Although $|K|$ can sometimes be reduced by domain-specific constraints (Section 3.4.2), it is generally $O(|Q|^2)$. Because $|Q|$ grows linearly with the total length of the samples, this version of the merging algorithm is only feasible for small amounts of data.

An alternative approach is to process samples incrementally, and start merging after a small amount of new data has been incorporated. This keeps the number of states, and therefore the number of candidates, small. If learning is successful, the model will stop growing eventually and reach a configuration that accounts for all new samples, at which point no new merges are required. (Figure 3.14 shows a size profile during incremental merging in one of our applications.) The incremental character is also more appropriate in scenarios where data is inherently incomplete and an on-line learning algorithm is needed that continuously updates a working model.

Best-first merging (on-line version)

Let M_0 be the empty model. Let $i := 0$. Loop:

explaining it.

A. Get some new samples X_i and incorporate into the current model M_i.

B. Loop:

1. Compute a set of candidate merges K from among the states of model M_i.
2. For each candidate $k \in K$ compute the merged model $k(M_i)$ and its posterior probability $P(k(M_i)|X)$.
3. Let k^* be the merge that maximizes $P(k(M_i)|X)$. Then let $M_{i+1} := k^*(M_i)$.
4. If $P(M_{i+1}|X) < P(M_i|X)$, break from the loop.
5. Let $i := i + 1$.

C. If the data is exhausted, break from the loop and return M_i as the induced model.

Incremental merging might in principle produce results worse than the batch version since the evaluation step doesn't have as much data at its disposal. However, we didn't find this to be a significant disadvantage in practice. One can optimize the number of samples incorporated in each step A (the *batch size*) for overall speed. This requires balancing the gains due to smaller model size against the constant overhead of each execution of step B. The best value will depend on the data and how much merging is actually possible on each iteration; we found between 1 and 10 samples at a time to be good choices.

One has to be careful not to start merging with extremely small models, such as that resulting from incorporating only a few short samples. Many of the priors discussed earlier contain logarithmic terms that approach singularities ($\log 0$) in this case, which can produce poor results, usually by leading to extreme merging. That can easily be prevented by incorporating a larger number of samples (say, 10 to 20) before going on to the first merging step.

Further modifications to the simple best-first search strategy are discussed in Section 3.4.5.

3.4 Implementation Issues

In this section we elaborate on the implementation of the various steps in the generic HMM merging algorithm presented in Section 3.3.5.

3.4.1 Efficient sample incorporation

In the simplest case this step creates a dedicated state for each instance of a symbol in any of the samples in X. These states are chained with transitions of probability 1, such that a sample $x_1 \ldots x_\ell$ is generated by a state sequence $q_1, \ldots, q_\ell$. q_1 can be reached from q_I via a transition of probability $\frac{1}{|X|}$, where $|X|$ is the total number of samples. State q_ℓ connects to q_F with probability 1. All states q_i emit their corresponding output symbol x_i with probability 1.

In this way, repeated samples lead to multiple paths through the model, all generating the sample string. The total probability of a string x according to the initial model is thus $\frac{c(x)}{|X|}$, *i.e.*, the relative frequency of string x. It follows that the initial model constitutes a maximum likelihood model for the data X.

Note that corresponding states in equivalent paths can be merged without loss of model likelihood. This is generally what the merging loop does in its initial passes.

A trivial optimization at this stage is to avoid the initial multiplicity of paths and check for each new sample whether it is already accounted for by an existing path. If so, only the first transition probability has to be updated.

The idea of shortcutting the merging of samples into the existing model could be pursued further along the lines of Thomason & Granum (1986). Using an extension of the Viterbi algorithm, the new sample can be aligned with the existing model states, recruiting new states only where necessary. Such an alignment couldn't effect all the possible merges, *e.g.*, it wouldn't be able to generate loops, but it could further reduce the initial number of states in the model, thereby saving computation in subsequent steps.[6]

3.4.2 Computing candidate merges

The general case here is to examine all of the $\frac{1}{2}|\mathcal{Q}|(|\mathcal{Q}| - 1)$ possible pairs of states in the current model. The quadratic cost in the number of states explains the importance of the various strategies to keep the number of states in the initial model small.

We have explored various application specific strategies to narrow down the set of worthwhile candidates. For example, if the cost of a merge is usually dominated by the cost of merging the output distributions of the states involved, we might index states according to their emission characteristics and consider only pairs of states with similar outputs. This constraint can be removed later after all other merging possibilities have been exhausted. The resulting strategy (first merging only same-outputs states, followed by general merging) not only speeds up the algorithm, it is also generally a good heuristic in incremental merging to prevent premature merges that are likely to be assessed differently in the light of new data.

Sometimes hard knowledge about the target model structure can further constrain the search. For example, word models for speech recognition are usually not allowed to generate arbitrary repetitions of subsequences (see Section 3.6.2). All merges creating loops (perhaps excepting self-loops) can therefore be eliminated in this case.

3.4.3 Model evaluation using Viterbi paths

To find the posterior probability of a potential model, we need to evaluate the structural prior $P(M_S)$ and, depending on the goal of the maximization, either find the maximum posterior probability (MAP) estimates for the model parameters θ_M, or evaluate the integral for $P(X|M_S)$ given by equation (2.20).

MAP estimation of HMM parameters could be done using the Baum-Welch iterative reestimation (EM) method, by taking the Dirichlet prior into account in the reestimation step. However, this would

[6] This optimization is as yet unimplemented.

would require an EM iteration for each candidate model, taking time proportional to the number all samples incorporated into the model.

Evaluation of $P(X|M_S)$, on the other hand, has no obvious exact solution at all, as discussed in Section 2.5.7.

In both cases the problem is greatly simplified if we use the Viterbi approximation, *i.e.*, the assumption that the probability of any given sample is due primarily to a single generation path in the HMM (Section 3.2.3).

Likelihood computation The exact model likelihood relative to a dataset X is given by the product of the individual sample probabilities, each of which is given by equation (3.1).

$$\begin{aligned} P(X|M) &= \prod_{x \in X} P(x|M) \\ &= \prod_{x \in X} \sum_{q_1 \ldots q_\ell \in \mathcal{Q}^\ell} p(q_I \to q_1) p(q_1 \uparrow x_1) p(q_1 \to q_2) \ldots p(q_\ell \uparrow x_\ell) p(q_\ell \to q_F) \end{aligned}$$

where ℓ is the length of sample x and $q_1 \ldots q_\ell$ denotes a path through the HMM, given as a state sequence.

The Viterbi approximation implies replacing the inner summations by the terms with the largest contribution:

$$P(X|M) \approx \prod_{x \in X} \max_{q_1 \ldots q_\ell \in \mathcal{Q}^\ell} p(q_I \to q_1) p(q_1 \uparrow x_1) p(q_1 \to q_2) \ldots p(q_\ell \uparrow x_\ell) p(q_\ell \to q_F)$$

The terms in this expression can be conveniently grouped by states, leading to the form

$$P(X|M) \approx \prod_{q \in \mathcal{Q}} \left(\prod_{q' \in \mathcal{Q}} p(q \to q')^{c(q \to q')} \prod_{\sigma \in \Sigma} p(q \uparrow \sigma)^{c(q \uparrow \sigma)} \right) \tag{3.9}$$

where $c(q \to q')$ and $c(q \uparrow \sigma)$ are the total counts of transitions and emissions occurring along the Viterbi paths associated with the samples in X. We use the notation $c^{(q)}$ for the collection of Viterbi counts associated with state q, so the above can be expressed more concisely as

$$P(X|M) = \prod_{q \in \mathcal{Q}} P(c^{(q)}|M)$$

MAP parameter estimation To estimate approximate MAP parameter settings based on Viterbi path counts, the maximum-likelihood estimates as given by (3.2) and (3.3) are modified to include the 'virtual' samples provided by the Dirichlet priors:

$$\hat{p}(q \to q') = \frac{c(q \to q') + \alpha_t - 1}{\sum_{s \in \mathcal{Q}} [c(q \to s) + \alpha_t - 1]} \tag{3.10}$$

$$\hat{p}(q \uparrow \sigma) = \frac{c(q \uparrow \sigma) + \alpha_e - 1}{\sum_{\rho \in \Sigma} [c(q \uparrow \rho) + \alpha_e - 1]}. \tag{3.11}$$

The α's are the prior proportions associated with the Dirichlet distributions for transitions and emissions respectively, as given in equation (3.6). (These are here assumed to be uniform for simplicity, but need not be.)

Note that the summations in (3.10) and (3.11) are over the entire set of possible transitions and emissions, which corresponds to a broad parameter prior. These summations have to be restricted to the transitions and emissions in the current model structure for narrow parameter priors.

Structure posterior evaluation To implement model comparison based on the posterior probabilities of the HMM structures (Section 2.5.7) we need to approximate the integral

$$P(X|M_S) = \int_{\theta_M} P(\theta_M|M_S)P(X|M_S,\theta_M)d\theta_M$$

We will apply the usual Viterbi approximation to $P(X|M_S,\theta_M)$, and assume in addition that the Viterbi paths do not change as θ_M varies. This approximation will be grossly inaccurate for broad parameter priors, but seems reasonable for narrow priors, where the topology largely determines the Viterbi path. More importantly, we expect this approximation to introduce a systematic error that does not bias the evaluation metric for or against any particular model structure, especially since the models being compared have only small structural differences.

The Viterbi-based integral approximation can now be written as

$$P(X|M_S) \approx \int_{\theta_M} P(\theta_M|M_S) \prod_{x \in Q} P(V(x)|M_S,\theta_M)d\theta_M ,$$

$V(x)$ being the Viterbi path associated with x. The parameters θ_M can now be split into their parts by state, $\theta_M = (\theta_M^{(q_1)}, \ldots, \theta_M^{(q_N)})$, and the integral rewritten as

$$\begin{aligned} P(X|M_S) &\approx \int_{\theta_M^{(q_1)}} \cdots \int_{\theta_M^{(q_N)}} \left(\prod_{q \in Q} P(\theta_M^{(q)}|M_S) \prod_{q \in Q} P(c^{(q)}|M_S,\theta_M^{(q)}) \right) d\theta_M^{(q_1)} \ldots d\theta_M^{(q_N)} \\ &= \prod_{q \in Q} \int_{\theta_M^{(q)}} P(\theta_M^{(q)}) P(c^{(q)}|M_S,\theta_M^{(q)}) d\theta_M^{(q)} \qquad (3.12) \end{aligned}$$

The integrals in the second expression can be evaluated in closed form by instantiating the generic formula for Dirichlet priors given in (2.16).

Optimistic Viterbi path updating So far, the Viterbi approximation has allowed us to decompose each of the likelihood, estimation, posterior evaluation problems into a form that allows computation by parts organized around states. To take full advantage of this fact we also need a way to update the Viterbi counts $c^{(q)}$ efficiently during merging. In particular, we want to avoid having to reparse all incorporated samples using the merged model. The approach taken here is to update the Viterbi counts associated with each state optimistically, *i.e.*, assuming that merging preserves the Viterbi paths.

During initial model creation the Viterbi counts are initialized to one, corresponding to the one sample that each state was created for. (If initial states are shared among identical samples the initial counts are set to reflect the multiplicity of the samples.) Subsequently, when merging states q_1 and q_2, the corresponding counts are simply added and recorded as the counts for the new state. For example, given

$c(q_1 \to q')$ and $c(q_2 \to q')$ in the current model, the merged state q_3 would be assigned a count

$$c(q_3 \to q') = c(q_1 \to q') + c(q_2 \to q') \quad .$$

This is correct if all samples with Viterbi paths through the transitions $q_1 \to q$ and $q_2 \to q'$ retain their most likely paths in the merged model, simply replacing the merged states with q_3, and no other samples change their paths to include $q_3 \to q'$.

This path preservation assumption is not strictly true but holds most of the time, since the merges actually chosen are those that collapse states with similar distributions of transition and emission probabilities. The assumption can be easily tested, and the counts corrected, by reparsing the training data from time to time.

In an incremental model building scenario, where new samples are available in large number and incorporated one by one, interleaved with merging, one might not want to store all data seen in the past. In this case an exponentially decaying average of Viterbi counts can be kept instead. This has the effect that incorrect Viterbi counts will eventually fade away, being replaced by up-to-date counts obtained form parsing more recent data with the current model.

Incremental model evaluation Using the techniques described in the previous sections, the evaluation of a model variant due to merging is now possible in $O(|Q| + |\Sigma|)$ amortized time, instead of the $O((|Q| + |\Sigma|) \cdot |Q| \cdot |X|)$ using a naive implementation.

Before evaluating specific candidates for merging, we compute the contributions to the posterior probability by each state in the current model. The prior will usually depend on the total number of states of the model; it is set to the current number minus 1 in these computations, thereby accounting for the prospective merge. The total computation of these contributions is proportional to the number of states and transitions, *i.e.* $O((|Q| + |\Sigma|) \cdot |Q|)$. For each potential merge we then determine the parts of the model it affects; these are precisely the transitions and emissions from the merged states, as well as transitions into the merged states. The total number of HMM elements affected is at most $O(|Q| + |\Sigma|)$. For all priors considered here, as well as the likelihood computations of (3.9) and (3.12), the old quantities can be updated by subtracting off the terms corresponding to old model elements and adding in the terms for the merged HMM. (The computation is based on addition rather than multiplication since logarithms are used for simplicity and accuracy).[7]

Since the initial computation of state contributions is shared among all the $O(|Q|^2)$ potential merges, the amortized time per candidate will also be on the order $|Q| + |\Sigma|$. Note that this is a worst case cost that is not realized if the HMMs are sparse as is usual. If the number of transitions and emissions on each state is bounded by a constant, the computation will also require only constant time.

[7] The evaluation of (3.12) involves computing multidimensional Beta functions, which are given as products of Gamma functions, one for each transition or emission. Therefore the addition/subtraction scheme can be used for incremental computation here as well. In practice this may not be worth the implementation effort if the absolute computational expense is small.

3.4.4 Global prior weighting

As explained previously, the merging strategy trades off generalization for fit to the data. Generalization is driven by maximizing the prior contribution, whereas the data is fit by virtue of maximizing the likelihood. In practice it is convenient to have a single parameter which controls the balance between these two factors, and thereby controls when generalization should stop.

From the logarithmic version of Bayes' law (2.13) we obtain

$$\log P(M) + \log P(X|M)$$

as the quantity to be maximized. To obtain such a global control parameter for generalization we can modify this to include a *prior weight* λ:

$$\lambda \log P(M) + \log P(X|M) \tag{3.13}$$

For $\lambda > 1$ the algorithm will stop merging later, and earlier for $\lambda < 1$.

The global prior weight has an intuitive interpretation as the reciprocal of a 'data multiplier.' Since the absolute, constant scale of the expression in (3.13) is irrelevant to the maximization, we can multiply by $\frac{1}{\lambda}$ to get

$$\log P(M) + \frac{1}{\lambda} \log P(X|M) = \log P(M) + \log P(X|M)^{\frac{1}{\lambda}}$$

This corresponds to the posterior given the data X repeated $\frac{1}{\lambda}$ times. In other words, by lowering the prior weight we pretend to have more data from the same distribution than we actually observed, thereby decreasing the tendency for merging to generalize beyond the data. We will refer to the actual number of samples $|X|$ multiplied by $\frac{1}{\lambda}$ as the *effective sample size*. The quantity $\frac{1}{\lambda}$ is thus equivalent to the multiplier c used in Quinlan & Rivest (1989) to model the 'representativeness' of the data.

Global prior weighting is extremely useful in practice. A good value for λ can be found by trial and error for a given amount of data, by starting with a small value and increasing λ successively, while cross-validating or inspecting the results. At each stage the result of merging can be used as the initial model for the next stage, thereby avoiding duplication of work.

Besides as a global generalization control, λ was also found to be particularly helpful in counteracting one of the potential shortcomings of incremental merging. Since incremental merging has to make decisions based on a subset of the data, it is especially important to prevent overgeneralization during the early stages. We can adjust λ depending on the number of samples processed to always maintain a minimum effective sample size during incremental merging, thereby reducing the tendency to overgeneralize based on few samples. This principle implies a gradual increase of λ as more samples are incorporated. An application of this is reported in Section 3.6.1.

3.4.5 Search issues

Section 3.3.5 has described the two basic best-first search strategies, *i.e.*, batch versus incremental sample processing. Orthogonal to that choice are various extensions to the search method to help overcome local posterior probability maxima in the space of HMM structures constructed by successive merging operations.

By far the most common problem found in practice is that the stopping criterion is triggered too early, since a single merging step alone decreases the posterior model probability, although additional related steps might eventually increase it. This happens although in the vast majority of cases the first step is in the right direction. The straightforward solution to this problem is to add a 'lookahead' to the best-first strategy. The stopping criterion is modified to trigger only after a fixed number of steps > 1 have produced no improvement; merging still proceeds along the best-first path. Due to this, the lookahead depth does not entail an exponential increase in computation as a full tree search would. The only additional cost is the work performed by looking ahead in vain at the end of a merging sequence. That cost is amortized over several samples if incremental merging with a batch size > 1 is being used.

Best-first merging with lookahead has been our method of choice for almost all applications, using lookaheads between 2 and 5. However, we have also experimented with *beam search* strategies. In these, a *set* of working models is kept at each time, either limited in number (say, the top K scoring ones), or by the difference in score to the current best model. On each inner loop of the search algorithm, all current models are modified according to the possible merges, and among the pool thus generated the best ones according to the beam criterion are retained. (By including the unmerged models in the pool we get the effect of a lookahead.)

Some duplication of work results from the fact that different sequences of merges can lead to the same final HMM structure. To remove such gratuitous duplicates from the beam we attach a list of *disallowed merges* to each model, which is propagated from a model to its successors generated by merging. Multiple successors of the same model have the list extended so that later successors cannot produce identical results from simply permuting the merge sequence.

The resulting beam search version of our algorithm does indeed produce superior results on data that requires aligning long substrings of states, and where the quality of the alignment can only be evaluated after several coordinated merging steps. On the other hand, beam search is considerably more expensive than best-first search and may not be worth a marginal improvement.

All results in Section 3.6 were obtained using best-first search with lookahead. Nevertheless, improved search strategies and heuristics for merging remain an important problem for future research.

3.5 Related Work

Many of the ideas used in our approach to Bayesian HMM induction are not new by themselves, and can be found in similar forms in the vast literatures on grammar induction and statistical inference.

3.5.1 Non-probabilistic finite-state models

At the most basic level we have the concept of state merging, which is implicit in the notion of state equivalence classes, and as such is pervasively used in much of automata theory (Hopcroft & Ullman 1979). It has also been applied to the induction of non-probabilistic automata (Angluin & Smith 1983).

Still in the field of non-probabilistic automata induction, Tomita (1982) has used a simple hill-climbing procedure combined with a goodness measure based on positive/negative samples to search the space of possible models. This strategy is obviously similar in spirit to our best-first search method (which uses a probabilistic goodness criterion based on positive samples alone).

The incremental version of the merging algorithm, in which samples are incorporated into a preliminary model structure one at a time, is similar in spirit (but not in detail) to the automata learning algorithm proposed by Porat & Feldman (1991), which induces finite-state models from positive-only, lexicographically ordered samples.

3.5.2 Bayesian approaches

The Bayesian approach to grammatical inference goes back at least to Horning (1969), where a procedure is proposed for finding the grammar with highest posterior probability given the data, using an enumeration of all candidate models in order of decreasing prior probability. While this procedure can be proven to converge to the maximum posterior probability grammar after a finite number of steps, it was found to be impractical when applied to the induction of context-free grammars. Horning's approach can be applied to any enumerable grammatical domain, but there is no reason to believe that the simple enumerative approach would be feasible in any but the most restricted applications. The HMM merging approach can be seen as an attempt to make the Bayesian strategy workable by operating in a more data-driven manner, while sacrificing optimality of the result.

3.5.3 State splitting algorithms

Enumerative search for finding the best model structure is also used by Bell *et al.* (1990)[8] to find optimal text compression models (for given number of number of states), although they clearly state that this is not a feasible practical approach. They also suggest both state merging and splitting as ways of constructing model structure dynamically from data, although the former is dismissed as being too inefficient for their purposes.[9]

Although the underlying intuitions are very similar, there are some significant conceptual differences between our work and their compression-oriented approaches. First, the evaluation functions used are invariably entropy (i.e., likelihood) based, and there is no formalized notion of a trade-off between model

[8] Thanks to Fernando Pereira for pointing out this reference. It is amazing how much overlap, apparently without mutual knowledge, there is between the text compression field and probabilistic computational linguistics. For example, the problem of smoothing zero-probability estimates and the solutions using mixtures (Bahl *et al.* 1983) or back-off models (Katz 1987) all have almost perfect analogs in the various strategies for building code spaces for compression models.

[9] Bell *et al.* (1990) attribute the state merging idea to Evans (1971).

complexity and data fit. Second, since the finite-state models they investigate act as encoder/decoders of text they are *deterministic*, i.e., the current state and the next input symbol determine a unique next state (it follows that each string has a unique derivation). This constrains the model space and allows states to be identified with string suffixes, which is the basis of all their algorithms. Finally, the models have no end states since they are supposed to encode continuous text. This is actually a minor difference since we can view the end-of-sentence as a special symbol, so that the final state is simply one that is dedicated to emitting that special symbol.

Bell *et al.* (1990) suggest *state splitting* as a more efficient induction technique for adaptively finding a finite-state model structure. In this approach, states are successively duplicated and differentiated according to their preceding context, whenever such a move promises to help the prediction of the following symbol. Ron *et al.* (1994) give a reformulation and formal analysis of this idea in terms of an information-theoretic evaluation function.

Interestingly, Bell *et al.* (1990) show that such a state splitting strategy confines the power of the finite-state model to that of a *finite-context* model. In models of this type there is always a finite bound k, such that the last k preceding symbols uniquely determine the distribution of the next symbol. In other words, state-based models derived by this kind of splitting are essentially n-gram models with variable (but bounded) context. This restriction applies equally to the algorithm of Ron *et al.* (1994).

By contrast, consider the HMM depicted in Figure 3.2, which is used below as a benchmark model. It describes a language in which the context needed for correct prediction of the final symbol is unbounded. Such a model can be found without difficulty by simple best-first merging. The major advantage of splitting approach is that it is guaranteed to find the appropriate model if enough data is presented and if the target language is in fact finite-context.

3.5.4 Other probabilistic approaches

Another probabilistic approach to HMM structure induction similar to ours is described by Thomason & Granum (1986). The basic idea is to incrementally build a model structure by incorporating new samples using an extended form of Viterbi alignment. New samples are aligned to the existing model so as to maximize their likelihood, while allowing states to be inserted or deleted for alignment purposes. The procedure is limited to HMMs that have a left-to-right ordering of states, however; in particular, no loops are allowed. In a sense this approach can be seen as an approximation to Bayesian HMM merging for this special class of models. The approximation in this case is twofold: the likelihood (not the posterior) is maximized, and only the likelihood of a single sample (rather than the entire data set) is considered.

Haussler *et al.* (1992) apply HMMs trained by the Baum-Welch method to the problem of protein primary structure alignment. Their model structures are mostly of a fixed, linear form, but subject to limited modification by a heuristic that inserts states ('stretches' the model) or deletes states ('shrinks' the model) based on the estimated probabilities.

Somewhat surprisingly, the work by Brown *et al.* (1992) on the construction of class-based n-gram models for language modeling can also be viewed as a special case of HMM merging. A class-based n-gram grammar is easily represented as an HMM, with one state per class. Transition probabilities represent the conditional probabilities between classes, whereas emission probabilities correspond to the word distributions for each class (for $n > 2$, higher-order HMMs are required). The incremental word clustering algorithm given in (Brown *et al.* 1992) then becomes an instance of HMM merging, albeit one that is entirely based on likelihoods.[10]

3.6 Evaluation

We have evaluated the HMM merging algorithm experimentally in a series of applications. Such an evaluation is essential for a number of reasons:

- The simple priors used in our algorithm give it a general direction, but little specific guidance, or may actually be misleading in practical cases, given finite data.
- Even if we grant the appropriateness of the priors (and hence the optimality of the Bayesian inference procedure in its ideal form), the various approximations and simplifications incorporated in our implementation could jeopardize the result.
- Using real problems (and associated data), it has to be shown that HMM merging is a practical method, both in terms of results and regarding computational requirements.

We proceed in three stages. First, simple formal languages and artificially generated training samples are used to provide a proof-of-concept for the approach. Second, we turn to real, albeit abstracted data derived from the TIMIT speech database. Finally, we give a brief account of how HMM merging is embedded in an operational speech understanding system to provide multiple-pronunciation models for word recognition.[11]

3.6.1 Case studies of finite-state language induction

3.6.1.1 Methodology

In the first group of tests we performed with the merging algorithm, the objective was twofold: we wanted to assess empirically the basic soundness of the merging heuristic and the best-first search strategy, as well as to compare its structure finding abilities to the traditional Baum-Welch method.

To this end, we chose a number of relatively simple regular languages, produced stochastic versions of them, generated artificial corpora, and submitted the samples to both induction methods. The probability

[10] Furthermore, after becoming aware of their work, we realized that the scheme Brown *et al.* (1992) are using for efficient recomputation of likelihoods after merging is essentially the same as the one we were using for recomputing posteriors (subtracting old terms and adding new ones).

[11] All HMM drawings in this section were produced using an *ad hoc* algorithm that optimizes layout using best-first search based on a heuristic quality metric (no Bayesian principles whatsoever were involved). We apologize for not taking the time to hand-edit some of the more problematic results, but believe the quality to be sufficient for expository purposes.

distribution for a target language was generated by assigning uniform probabilities at all choice points in a given HMM topology.

The induced models were compared using a variety of techniques. A simple quantitative comparison is obtained by computing the log likelihood on a test set. This is proportional to the negative of (the empirical estimate of) the cross-entropy, which reaches a minimum when the two distributions are identical.

To evaluate the HMM topology induced by Baum-Welch training, the resulting models are *pruned*, *i.e.*, transitions and emissions with probability close to zero are deleted. The resulting structure can then be compared with that of the target model, or one generated by merging. The pruning criterion used throughout was that a transition or emission had an expected count of less than 10^{-3} given the training set.

Specifically, we would like to check that the resulting model generates exactly the same discrete language as the target model. This can be done empirically (with arbitrarily high accuracy) using a simple Monte-Carlo experiment. First, the target model is used to generate a reasonably large number of samples, which are then parsed using the HMM under evaluation. Samples which cannot be parsed indicate that the induction process has produced a model that is not sufficiently general. This can be interpreted as *overfitting* the training data.

Conversely, we can generate samples from the HMM in question and check that these can be parsed by the target model. If not, the induction has *overgeneralized*.

In some cases we also inspected the resulting HMM structures, mostly to gain an intuition for the possible ways in which things can go wrong. Some examples of this are presented below.

Note that the outcome of the Baum-Welch algorithm may (and indeed does, in our experience) vary greatly with the initial parameter settings. We therefore set the initial transition and emission probabilities randomly from a uniform distribution, and repeated each Baum-Welch experiment 10 times. Merging, on the other hand, is deterministic, so for each group of runs only a single merging experiment was performed for comparison purposes.

Another source of variation in the Baum-Welch method is the fixed total number of model parameters. In our case, the HMMs are fully parameterized for a given number of states and set of possible emissions; so the number of parameters can be simply characterized by the number of states in the HMM.[12]

In the experiments reported here we tried two variants: one in which the number of states was equal to that of the target model (the minimal number of states for the language at hand), and a second one in which additional states were provided.

Finally, the nature of the training samples was also varied. Besides a standard random sample from the target distribution we also experimented with a 'minimal' selection of representative samples, chosen to be characteristic of the HMM topology in question. This selection is heuristic and can be characterized as follows: "From the list of all strings generated by the target model, pick a minimal subset, in order of decreasing probability, such that each transition and emission in the target model is exercised at least once, and such that each looping transition is exemplified by a sample that traverses the loop at least twice." The minimal

[12] By convention, we exclude initial and final states in these counts.

training samples this rule produces are listed below, and do in fact seem to be intuitively representative of their respective target models.

3.6.1.2 Priors and merging strategy

A uniform strategy and associated parameter settings were used in all merging experiments.

The straightforward description length prior for HMM topologies from Section 3.3.3.3, along with a narrow Dirichlet prior for the parameters (Section 3.3.3.2) were used to drive generalization. The total Dirichlet prior weight α_0 for each multinomial was held constant at $\alpha_0 = 1$, which biases the probabilities to be non-uniform (in spite of the target models used). The objective function in the maximization was the posterior of the HMM structure, as discussed in Section 2.5.7.

Merging proceeded using the incremental strategy described in Section 3.3.5 (with batch size 1), along with several of the techniques discussed earlier. Specifically, incremental merging was constrained to be among states with identical emissions at first, followed by an unconstrained batch merging phase. The global prior weight λ was adjusted so as to keep the effective sample size constant at 50. In accordance with the rationale given in Section 3.4.4, this gradually increases the prior weight during the incremental merging phase, thereby preventing early overgeneralization.

The search strategy was best-first with 5 steps lookahead.

3.6.1.3 Case study I

The first test task was to learn the regular language $ac^*a \cup bc^*b$, generated by the HMM in Figure 3.2.[13] It turns out that the key difficulty in this case is to find the dependency between first and last symbols, which can be separated by arbitrarily long sequences of intervening (and non-distinguishing) symbols.[14]

The minimal training sample used for this model consisted of 8 strings:

aa
bb
aca
bcb
$acca$
$bccb$
$accca$
$bcccb$

Alternatively, a sample of 20 random strings was used.

The results of both the merging and the Baum-Welch runs are summarized by the series of plots in Figure 3.3. The plots in the left column refer to the minimal training sample runs, whereas the right column

[13] We use standard regular expression notation to describe finite-state languages: x^k stands for k repetitions of the string x, x^* denotes 0 or more repetitions of x, x^+ stands for 1 or more repetitions, and $\cup$ is the disjunction (set union) operator.

[14] This test model was inspired by finite-state models with similar characteristics that have been the subject of investigations into human language learning capabilities (Reber 1969; Cleeremans 1991)

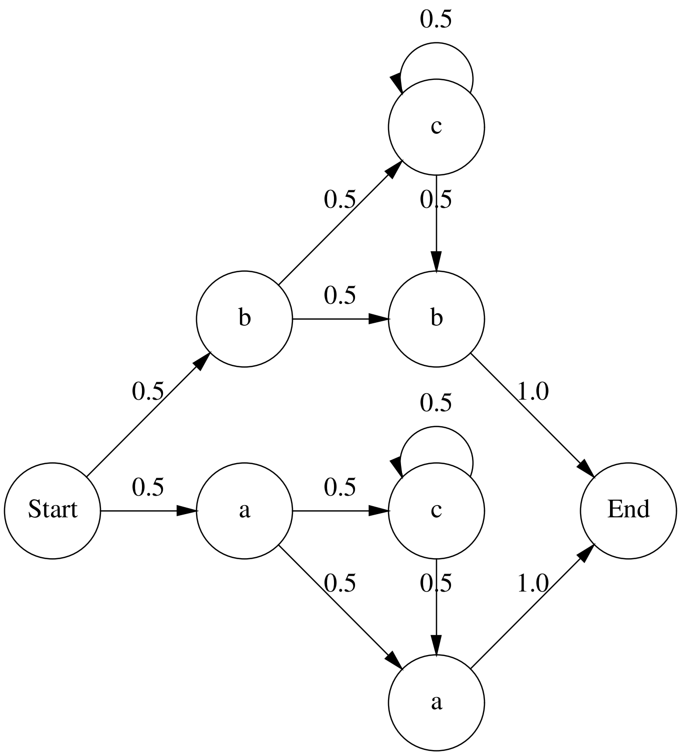


Figure 3.2: Case study I: HMM generating the test language $ac^*a \cup bc^*b$.

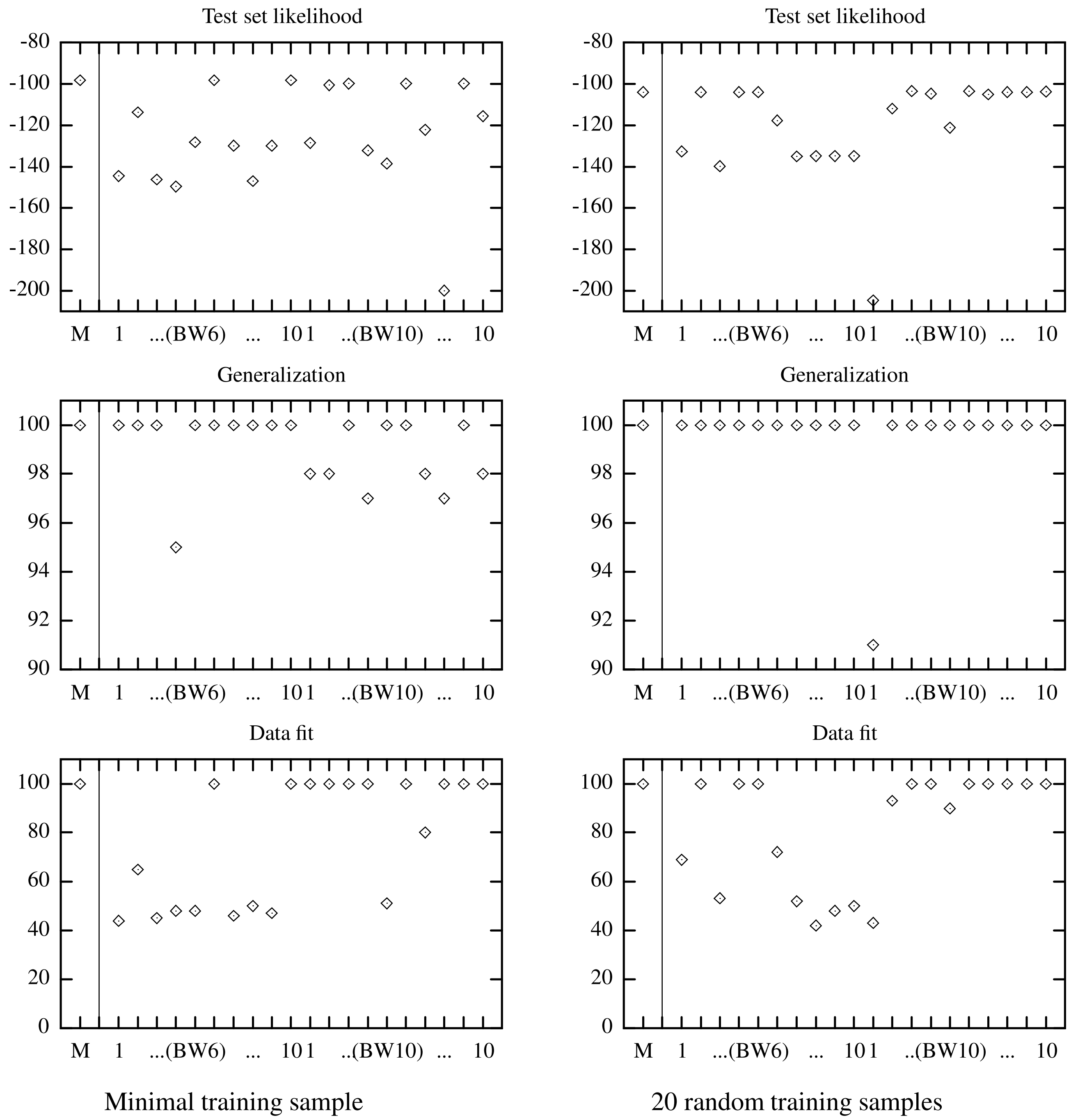


Figure 3.3: Case study I: Results of induction runs.

shows the corresponding data for the random 20 string sample runs. Each plot shows a quantitative measure of the induced models' performance, such that the x-axis represents the various experiments. In each case, the left-most data point (to the left of the vertical bar) represents the single merging (M) run, followed by 10 data points for repeated Baum-Welch runs with minimal number of states (BW6), and 10 more data points for Baum-Welch runs with 10 states (BW10).

The top row plots the log likelihood on a 100 sample test set; the higher the value, the lower the relative entropy (Kullback-Leibler distance) between the distribution of strings generated by the target model and that embodied by the induced model. To obtain comparable numbers, the probabilities in both the merged model and those estimated by Baum-Welch are set to their ML estimates (ignoring the parameter prior used during merging).

The second row of plots shows the results of parsing the same 100 samples using the discretized induced model topologies, *i.e.*, the number of samples successfully parsed. A score of less than 100 means that the induced model is too specific.

The third row of plots shows the converse parsing experiment: how many out of 100 random samples generated by each induced model can be parsed by the target model. (Note that these 100 samples therefore are not the same across runs.) Therefore, a score of less than 100 indicates that the induced model is overly general.

Note that we use the terms 'general' and 'specific' in a loose sense here which includes cases where two models are not comparable in the set-theoretic sense. In particular, a model can be both 'more general' and 'more specific' than the target model.

When evaluating the structural properties of a model we consider as a 'success' those which neither overgeneralize nor overfit. Such models invariably also have a log likelihood close to optimal. The log likelihood alone, however, can be deceptive, *i.e.*, it may appear close to optimal even though the model structure represents poor generalization. This is because the critical, longer samples that would be indicative of generalization have small probability and contribute little to the average log likelihood. This was the primary reason for devising the parsing experiments as an additional evaluation criterion.

Results The merging procedure was able to find the target model structure for both types of training sets. The left-most data points in the plots can therefore be taken as benchmarks in evaluating the performance of the Baum-Welch method on this data.

The quality of the Baum-Welch induced model structures seems to vary wildly with the choice of initial conditions. For the minimal sample, 2 out of 10 runs resulted in perfect model structures when working with 6 states; 3 out of 10 when using 10 states (four more than necessary). When given a random training sample instead, the success rate improved to 3/10 and 7/10, respectively.

The overgeneralizations observed in Baum-Welch derived models correspond mostly to a missing correlation between initial and final symbols. These models typically generate some subset of $(a \cup b)c^*(a \cup b)$ which leads to about 50% of the samples generated to be rejected by the target model (cf. bottom plots).

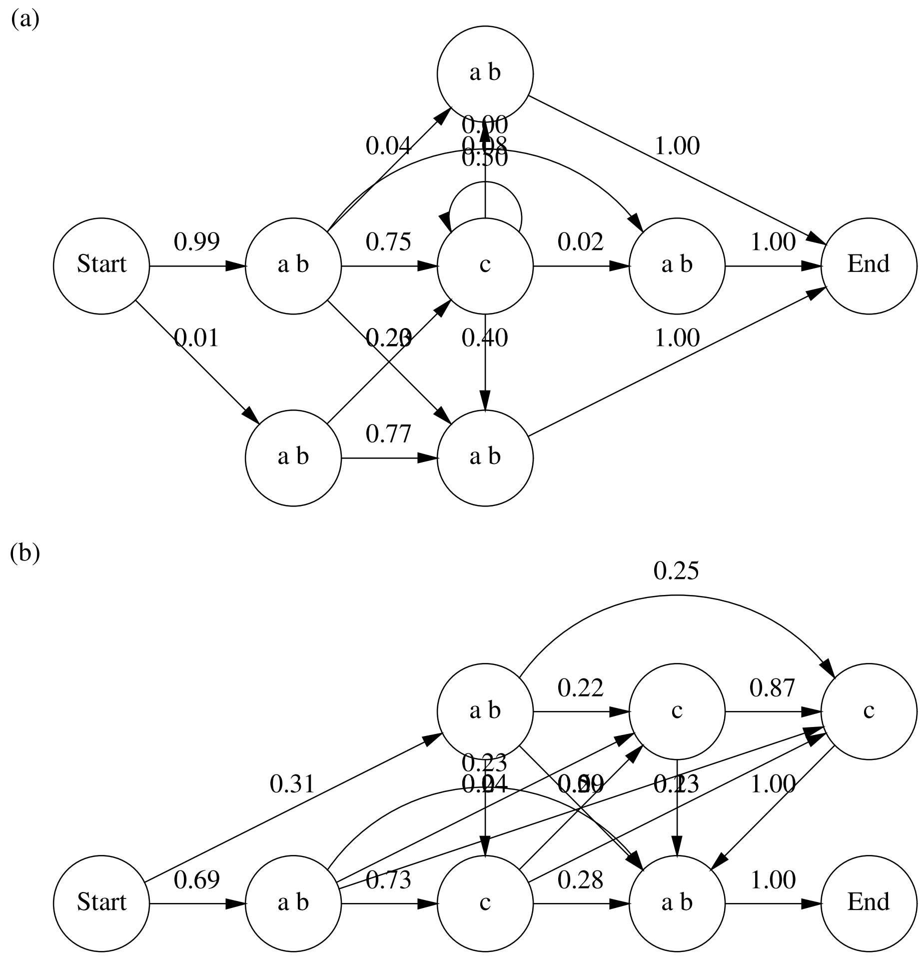


Figure 3.4: Case study I: BW-derived HMM structures that fail on generalization.

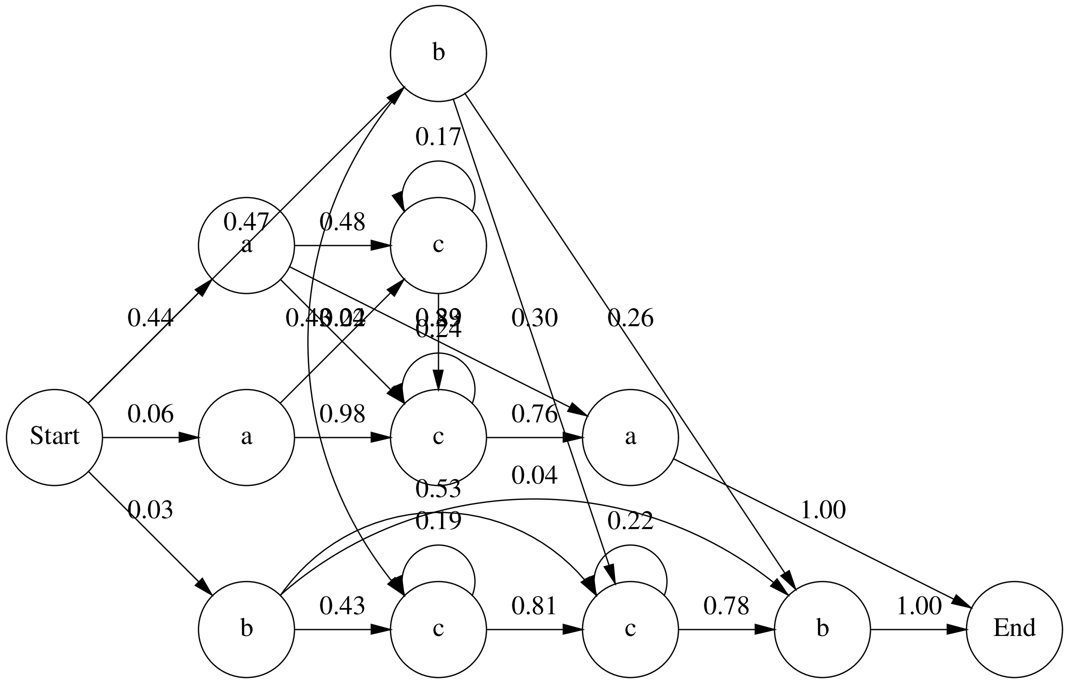


Figure 3.5: Case study I: Redundant BW-derived HMM structure for $ac^*a \cup bc^*b$.

Baum-Welch studies It is instructive to inspect some of the HMM topologies found by the Baum-Welch estimator. Figure 3.4 shows models of 6 states trained on minimal samples, one exhibiting overgeneralization, and one demonstrating both overfitting and overgeneralization.

The HMM in (a) generates $(a \cup b)c^*(a \cup b)$ and has redundantly allocated states to generate $a \cup b$. The HMM in (b) generates $(a \cup b)c^k(a \cup b)$, for $k = 0, 1, 2, 3$. Here, precious states have been wasted modeling the repetition of c's, instead of generalizing to a loop over a single state and using those states to model the distinction between a and b.

If estimation using the minimal number of states (6 in this case) is successful, the discretized structure invariably is that of the target model (Figure 3.2), as expected, although the probabilities will depend on the training sample used. Successful induction using 10 states, on the other hand, leads to models that, by definition, contain redundant states. However, the redundancy is not necessarily a simple duplication of states found in the target model structure. Instead, rather convoluted structures are found, such as the one in Figure 3.5 (induced from the random 20 samples).

Merging studies We also investigated how the merging algorithm behaves for non-optimal values of the global prior weight λ. As explained earlier, this value is implicit in the number of 'effective' samples, the parameter that was maintained constant in all experiments, and which seems to be robust over roughly an order of magnitude.

We therefore took the resulting λ value and adjusted it both upward and downward by an order of

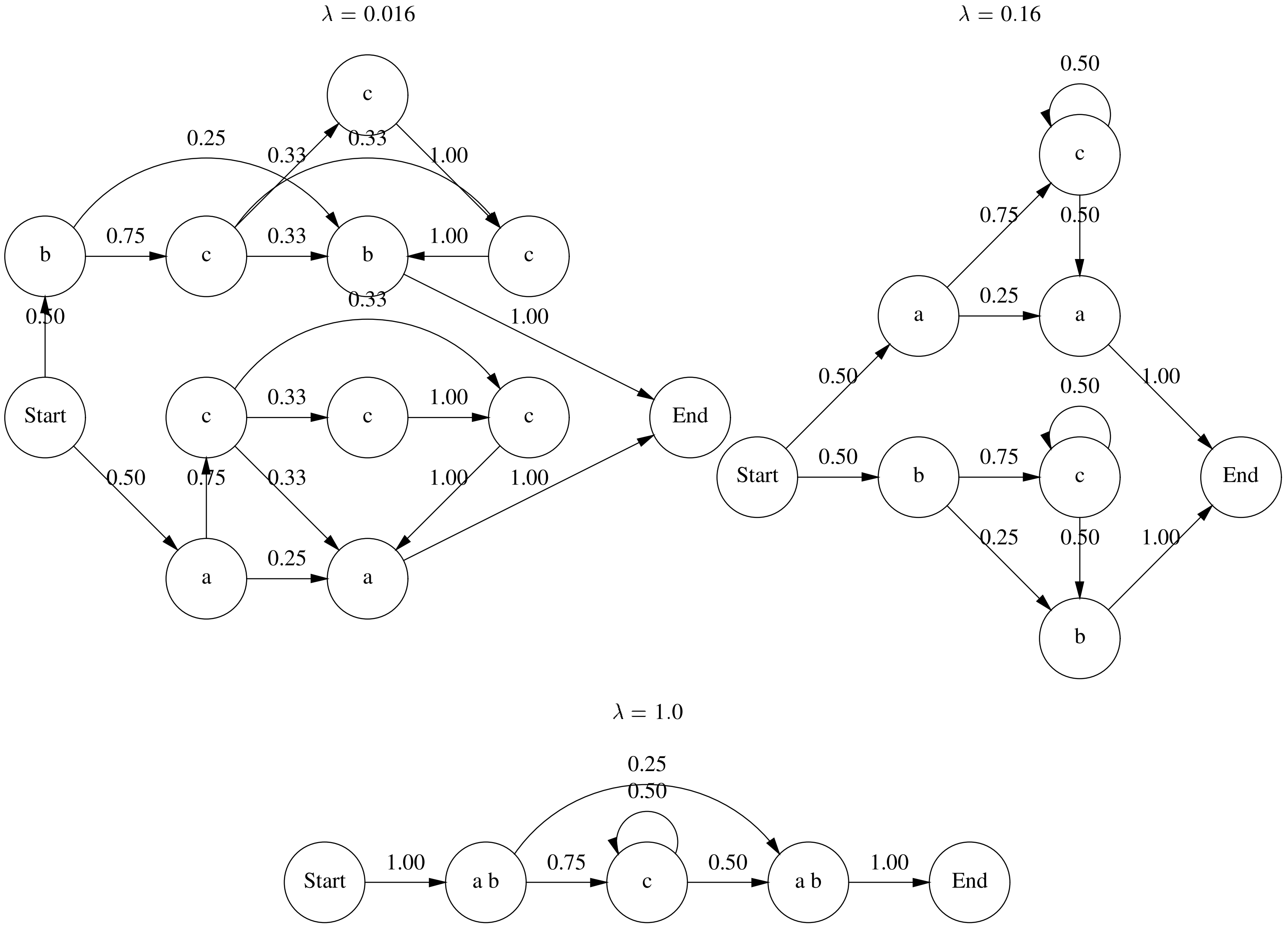


Figure 3.6: Case study I: Generalization depending on global prior weight.

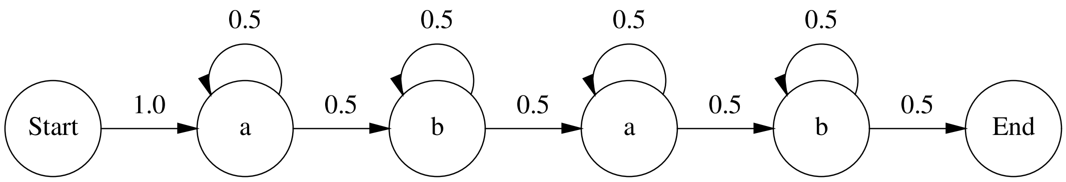


Figure 3.7: Case study II: HMM generating the test language $a^+b^+a^+b^+$.

magnitude to produce undergeneralized (overfitted) and overgeneralized models, respectively. The series of models found (using the minimal sample) is shown in Figure 3.6.

For $\lambda = 0.016$ no structural generalization takes place; the sample set is simply represented in a concise manner. For a wide range around $\lambda = 0.16$, the target HMM is derived, up to different probability parameters. A further increase to $\lambda = 1.0$ produces a model whose structure no longer distinguishes between a and b. One could argue that this overgeneralization is a 'natural' one given the data.

3.6.1.4 Case study II

The second test language is $a^+b^+a^+b^+$, generated by the HMM depicted in Figure 3.7. The minimal training sample contains the following nine strings

$abab$
$aabab$
$abbab$
$abaab$
$ababb$
$aaabab$
$abbbab$
$abaaab$
$ababbb$

The other training sample once again consisted of 20 randomly drawn strings.

Figure 3.8 presents the results in graphical form, using the same measures and arrangement as in the previous case study. (However, note that the ranges on some of the y-axes differ.)

Similar to the previous experiment, the merging procedure was successful in finding the target model, whereas the Baum-Welch estimator produced inconsistent results that were highly dependent on the initial parameter settings. Furthermore, the Baum-Welch success rates seemed to reverse when switching from the minimal to the random sample (from 6/10 and 0/10 to 1/10 and 6/10, respectively). This is disturbing since it reveals a sensitivity not only to the number of states in the model, but also to the precise statistics of the sample data.

The overgeneralizations are typically of the form $(a^+b^+)^k$, where either $k = 1$ or $k > 2$.

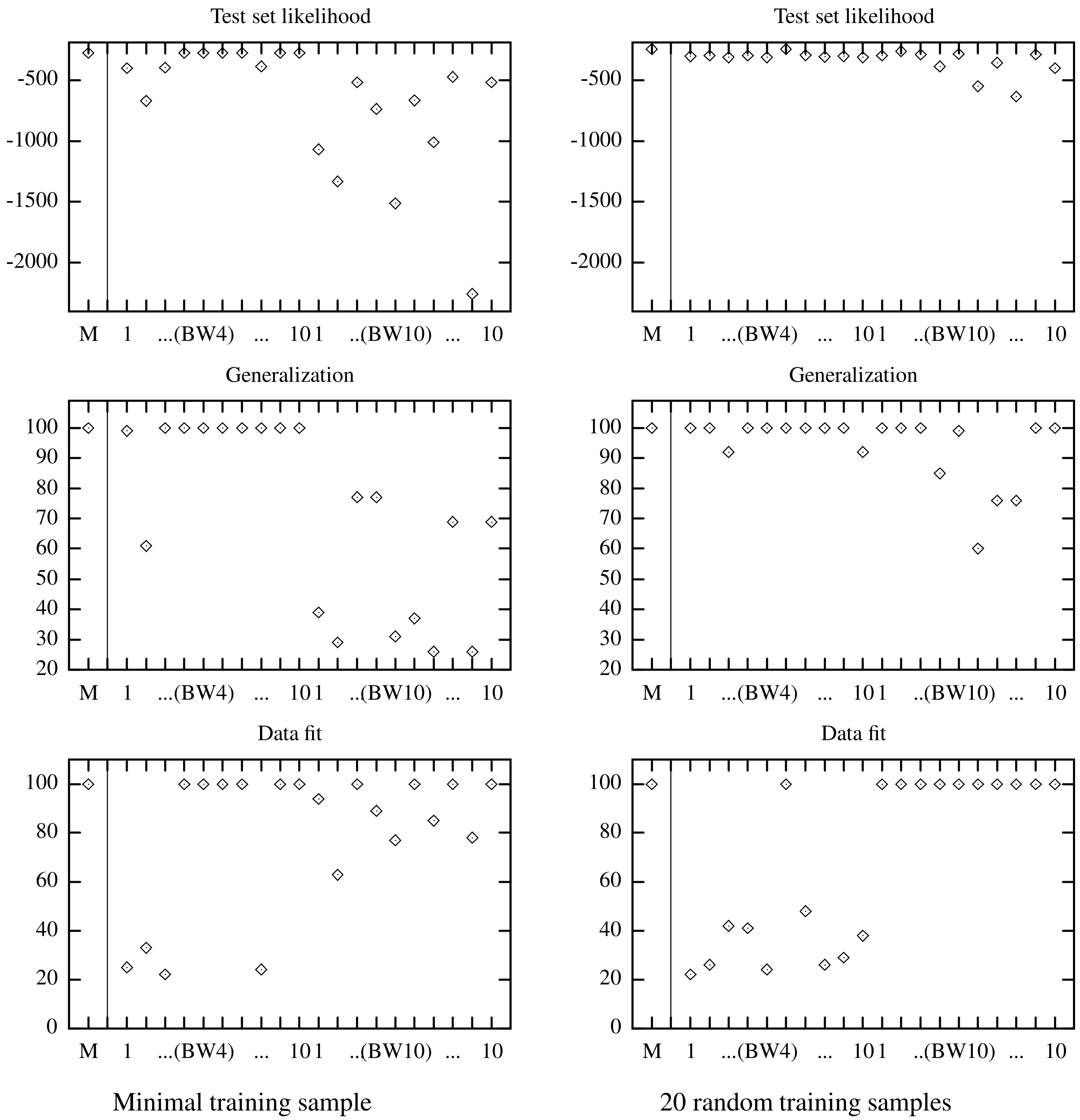
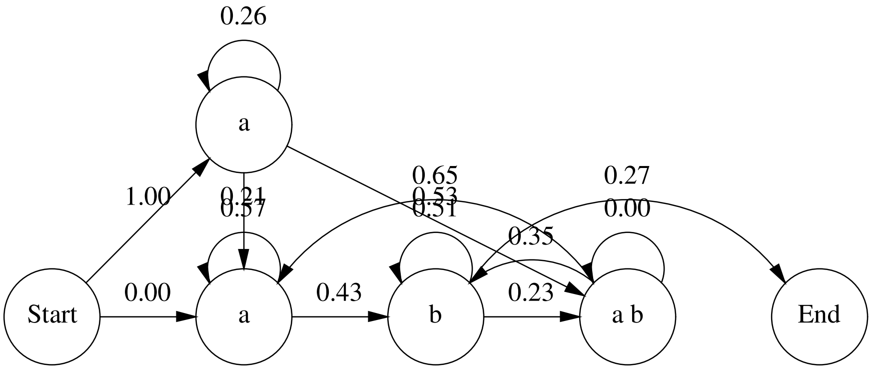


Figure 3.8: Case study II: Results of induction runs.

(a)



(b)

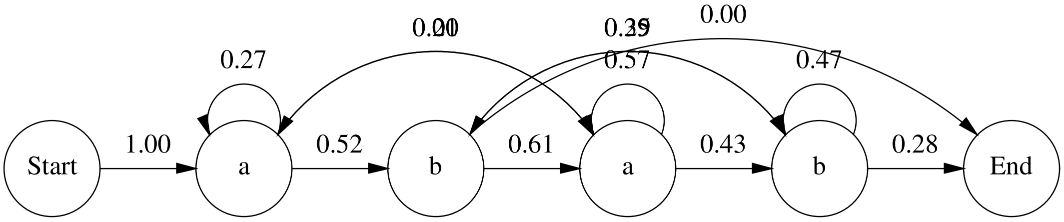


Figure 3.9: Case study II: BW-derived HMM structures that fail on generalization.

Baum-Welch studies As in the previous case study, we looked at various model structures found by Baum-Welch estimation. All examples in this section are from training on the 20 random samples.

Figure 3.9(a) shows a structure that is overly general: it generates $(a \cup a^+ b^+)(a \cup b)^+ b^+$. In (b), we have an HMM that partly overgeneralizes, but at the same time exhibits a rather peculiar case of overfitting: it excludes strings of the form $a^+ b^k a^+ b^+$ where k is even. No such cases happened to be present in the training set.

The accurate model structures of 10 states found by the Baum-Welch method again tended to be rather convoluted. Figure 3.10 shows as case in point.

Merging studies We also repeated the experiment examining the levels of generalization by the merging algorithm, as the value of the global prior weight was increased over three orders of magnitude.

Figure 3.11 shows the progression of models for $\lambda = 0.018, 0.18$, and 1.0. The pattern is similar to that in in the first case study (Figure 3.11). The resulting models range from a simple merged representation of the samples to a plausible overgeneralization from the training data ($(a^+ b^+)^+$). The target model is obtained for λ values between these two extremes.

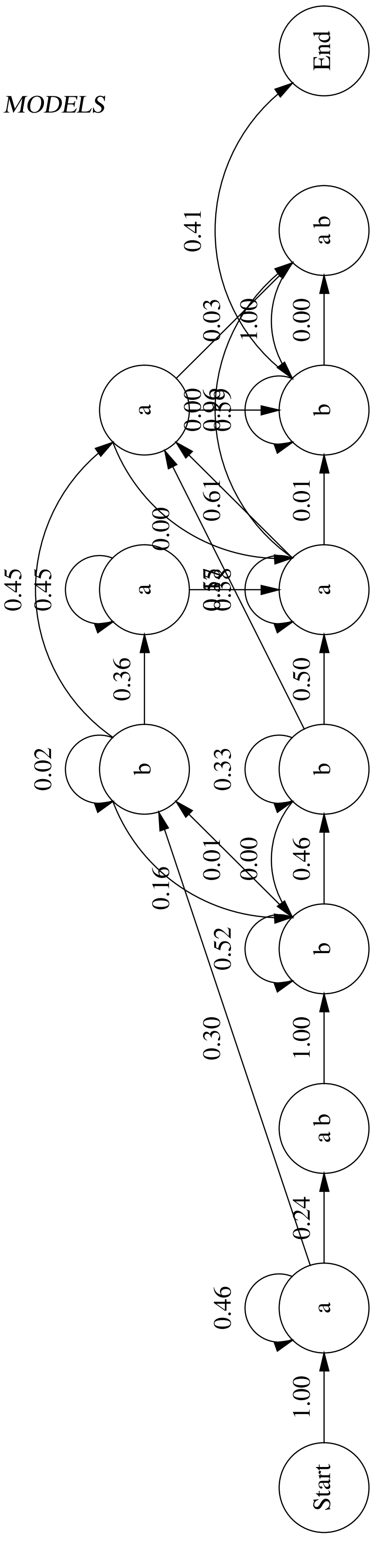
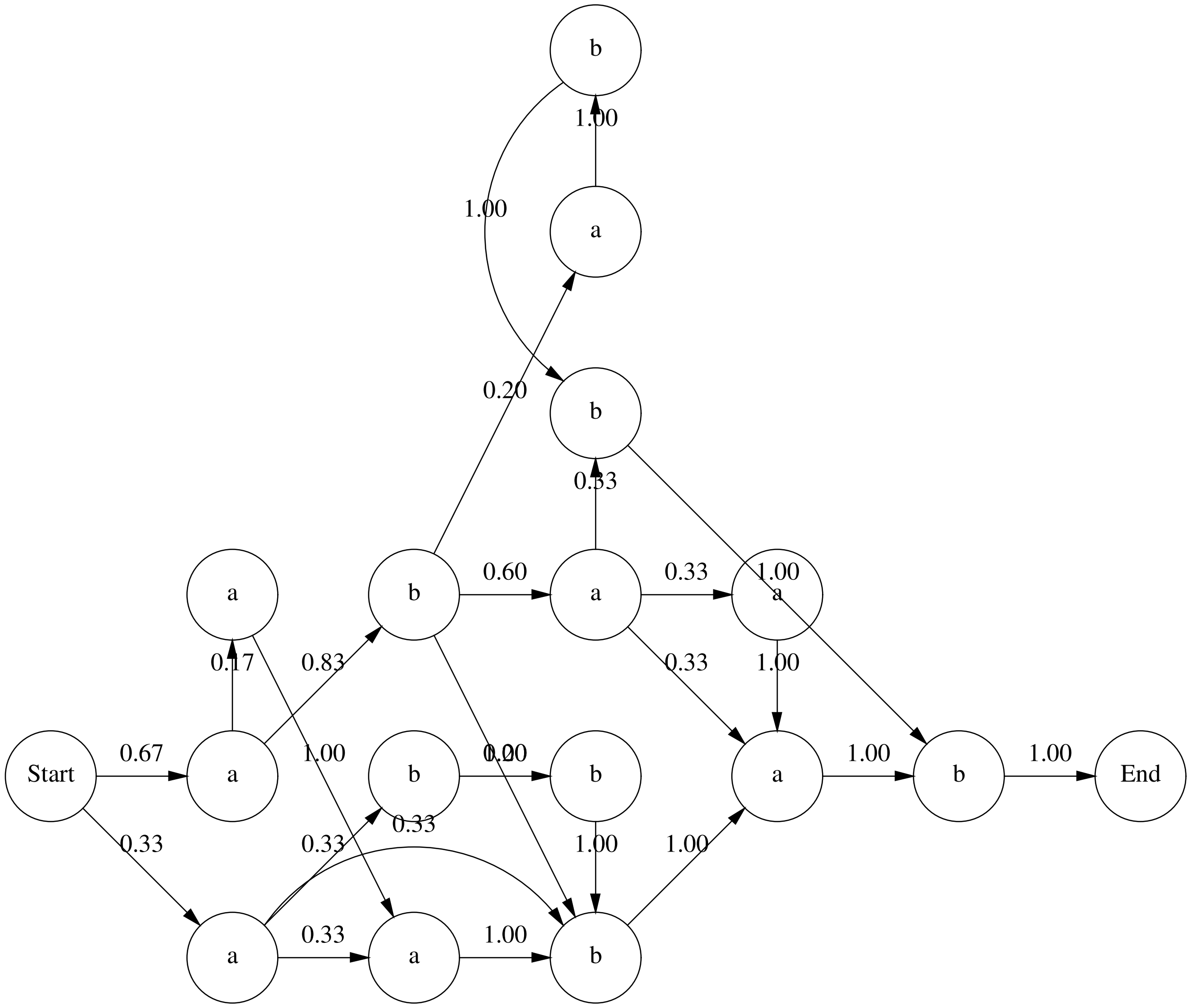
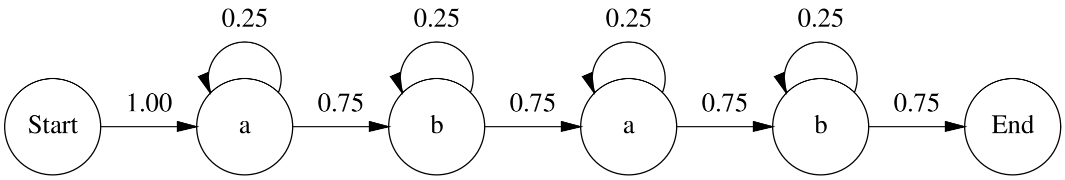


Figure 3.10: Case study II: Redundant BW-derived HMM structure for $a^+b^+a^+b^+$.

$\lambda = 0.018$



$\lambda = 0.18$



$\lambda = 1.0$

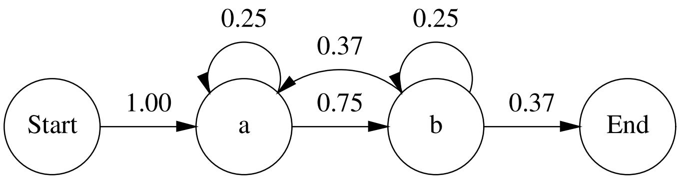


Figure 3.11: Case study II: Generalization depending on global prior weight.

3.6.1.5 Discussion

It is tempting to try to find a pattern in the performance of the Baum-Welch estimator in terms of parameters such as model size and sample size and type (although this would go beyond the intended scope of this study). Regarding model size, one would expect the smaller minimal models to produce better coverage of the target language, and a tendency to overgeneralize, since too few states are available to produce a close fit to the data. This is indeed observable in the plots in the bottom rows of Figures 3.3 and 3.8: the runs in the left half typically produce a higher number of rejected (overgeneralized) samples.

Conversely, one expects a greater tendency toward overfitting in the training runs using more than the minimal number of states. The plots in the middle rows of Figures 3.3 and 3.8 confirm this expectation: the right halfs show a greater number of rejected strings from the target language, indicating insufficient generalization.

It is conceivable that for each language there exists a model size that would lead to a good compromise between generalization and data fit so as to produce reliable structure estimation. The problem is that there seems to be no good way to predict that optimal size.[15]

Successful use of the model merging approach also relies on suitable parameter choices, mainly of the global prior weight (or the number of 'effective samples'). The prime advantage of merging in this regard is that the parameters seem to be more robust to both sample size and distribution, and the mechanics of the algorithm make it straightforward to experiment with them. Furthermore, it appears that overgeneralization by excessive merging tends to produce 'plausible' models (with the obvious caveat that this conclusion is both tentative given the limited scope of the investigation, and a matter of human judgment).

3.6.2 Phonetic word models from labeled speech

3.6.2.1 The TIMIT database

In the second evaluation stage, we were looking for a sizeable collection of real-world data suitable for HMM modeling. The TIMIT (Texas Instruments-MIT) database is a collection of hand-labeled speech samples compiled for the purpose of training speaker-independent phonetic recognition systems (Garofolo 1988). It contains acoustic data segmented by words and aligned with discrete labels from an alphabet of 62 phones. For our purposes, we ignored the continuous, acoustic data and viewed the database simply as a collection of string samples over a discrete alphabet.

The goal is to construct a probabilistic model for each word in the database, representing its phonetic structure as accurately as possible, *i.e.*, maximizing the probabilities of the observed pronunciations. A fraction of the total available data, the *test set*, is set aside for evaluating the induced models, while the rest is used to induce or otherwise estimate the probabilistic model for each word. By comparing the performance of the models generated by various methods, along with other relevant properties such as model size, processing time, *etc.*, we can arrive at a reasonably objective comparison of the various methods. Of course, the ultimate

[15] In Section 3.6.2 we actually use a simple heuristic that scales the model sizes linearly with the length of the samples. This heuristic works rather well in that particular application, but it crucially relies on the models being loop-free, and hence wouldn't apply generally.

test is to use pronunciation models in an actual system that handles acoustic data, a considerably more involved task. In Section 3.6.3 we will describe such a system and how the HMM merging process was brought to bear on it.

The full TIMIT dataset consists of 53355 phonetic samples for 6100 words.[16] To keep the task somewhat manageable, and to eliminate a large number of words with too few samples to allow meaningful structural model induction, we used a subset of this data consisting of words of intermediate frequency. Arbitrarily, we included words occurring between 20 and 100 times in the dataset. This left us with a working dataset of 206 words, comprising a total of 7861 samples. Of these, 75% for each word (5966 total) were made available to various training methods, while the remaining 25% (1895 total) were left for evaluation.

3.6.2.2 Qualitative evaluation

In preliminary work, while evaluating the possibility of incorporating HMM merging in an ongoing speech understanding project, Gary Tajchman, a researcher at ICSI with extensive experience in acoustic-phonetic modeling, experimented with the algorithm using the TIMIT database. He inspected a large number of the resulting models for 'phonetic plausibility', and found that they generally appeared sound, in that the generated structures were close to conventional linguistic wisdom.

To get an idea for the kinds of models that HMM merging produces from this data it is useful to examine an example. Figure 3.13 shows an HMM constructed from 37 samples of the word *often*. For comparison, Figure 3.12 shows the initial HMM constructed from the samples, before merging (but with identical paths collapsed).

Figure 3.14 plots the number of states obtained during on-line (incremental) merging as a function of the number of incorporated samples. The numbers of states before and after merging at each stage are plotted as adjacent datapoint, giving rise to the spikes in the figure. Merging starts with five incorporated samples (17 states). Initially merging occurs with almost every additional sample, but later on most samples are already parsed by the HMM and require no further merging.

The most striking feature of the induced HMM is that both the first and the second syllable of *often* contain a number of alternative pronunciations anchored around the central [f] consonant, which is common to all variants. This structural property is well mirrored in the two branching sections of the HMM. A number of the pronunciation for the second syllable share the optional [tcl t] sequence.

Notice that each state (except initial and final) has exactly one output symbol. This constraint was imposed due to the particular task we had in mind for the resulting HMMs. The speech recognition system for which these word models are intended implicitly assumes that each HMM state represents a unique phone. Such a restriction can be easily enforced in the algorithm by filtering the merge candidates for pairs of states with identical emissions.

The single-output constraint does not limit the representational power of the HMMs, since a multi-output state can always be split into several single-output states. It does, however, affect the structural prior

[16] This is the union of the 'training' and 'test' portions in the original TIMIT distribution.

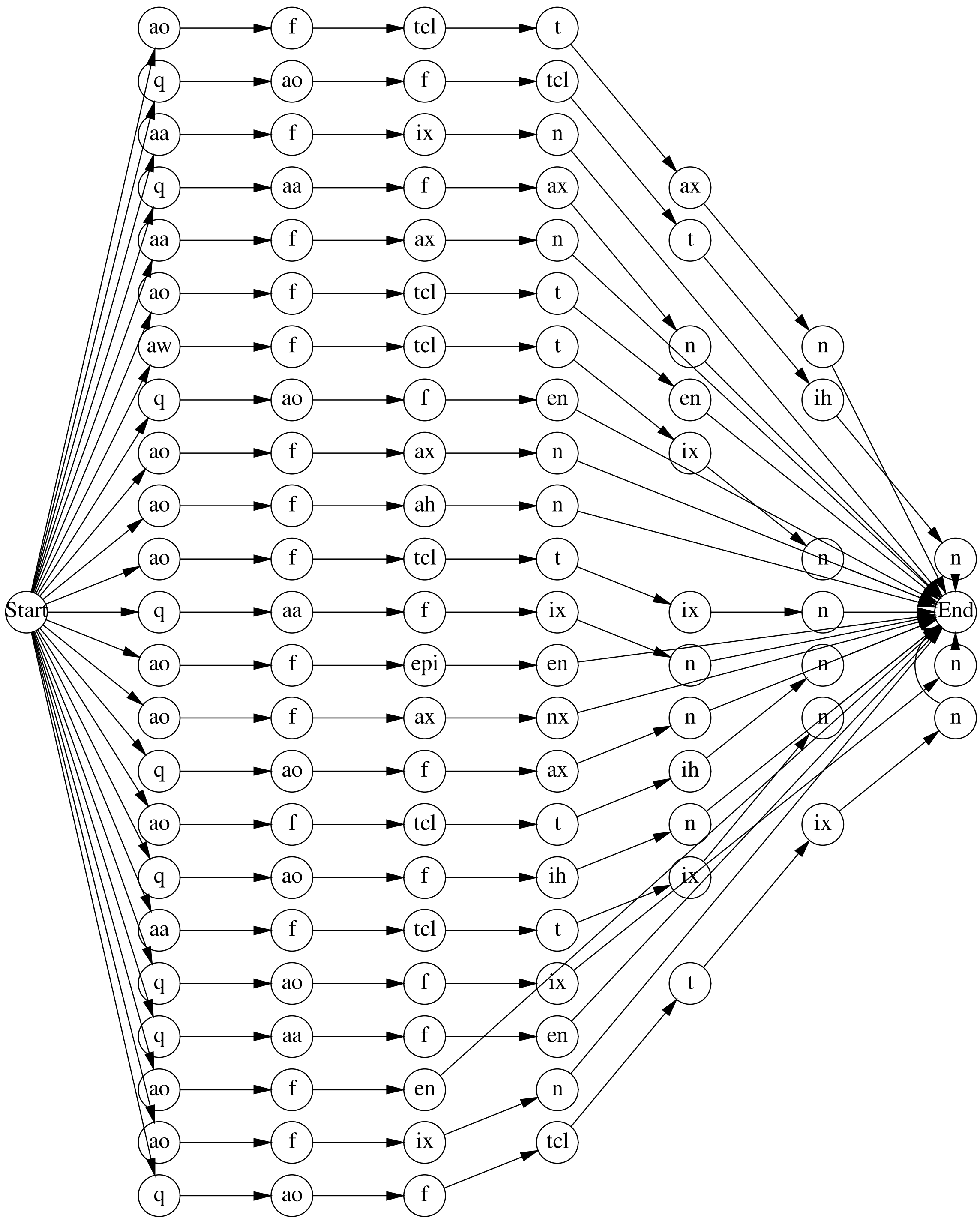


Figure 3.12: Initial HMM constructed from 37 samples of the word *often*.

Probabilities are omitted in this graph. Due to repetitions in the data the HMM has only 23 distinct paths.

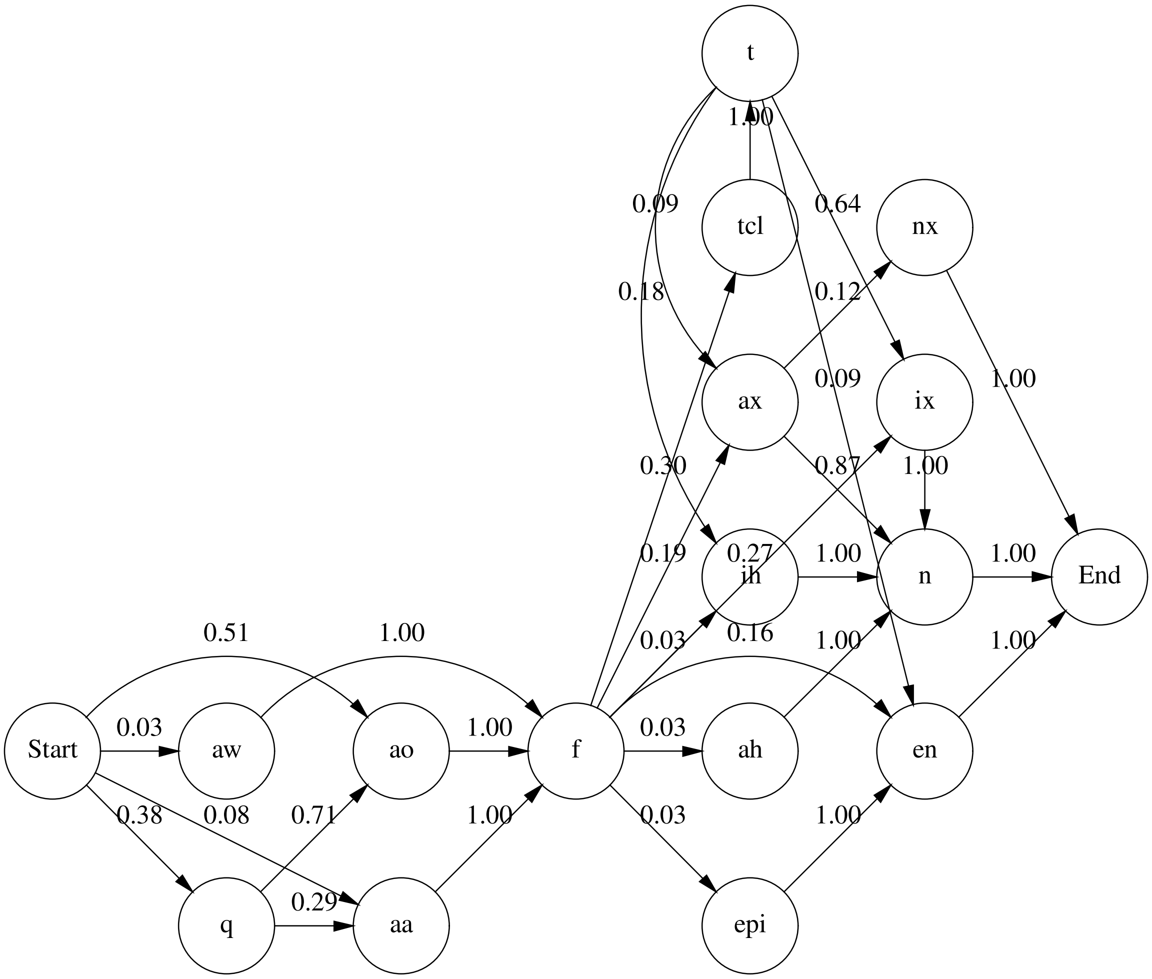


Figure 3.13: Merged HMM constructed from 37 samples of the word *often*.

Merging was constrained to keep the emission on each state unique.

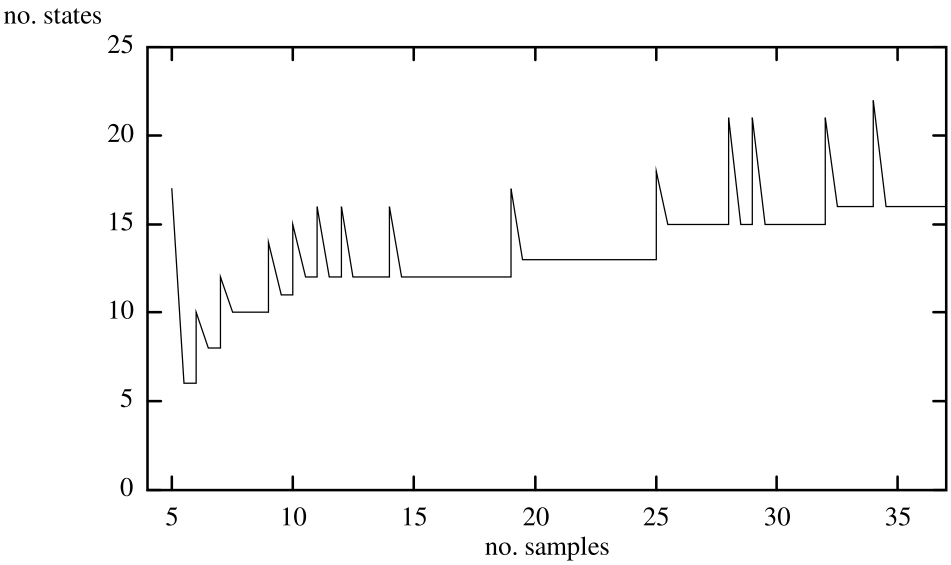


Figure 3.14: Number of HMM states as a function of the number of samples incorporated during incremental merging.

> Each spike represents the states added to model an unparseable sample, which are then (partly) merged into the existing HMM structure.

for the HMM. In a single-output HMM, each emission carries a prior probability of $\frac{1}{|\Sigma|}$, rather than one of the various structural priors over multinomials discussed in Section 3.3.3.3. Incidentally, the constraint can also speed up the algorithm significantly, because many candidates are efficiently eliminated that would otherwise have to be evaluated and rejected. This advantage by far outweighs the larger size of the derived models.

A second constraint can be enforced in this particular domain (and possibly others). Since the resulting HMMs are meant to be phonetic word models, it does not make sense to allow loops in these models. In very rare circumstances the merging algorithm might be tempted to introduce such loops, *e.g.*, because of some peculiar repetitive pattern in a sample ('banana'). Given our prior knowledge in this regard we can simply rule out candidate merges that would introduce loops.[17]

HMM merging can thus be used to derive models for allophonic variation from data, without explicitly representing a mapping from individual phonemes to their realizations. This is in contrast to other approaches where one first induces rules for pronunciations of individual phonemes based on their contexts (*e.g.*, using decision tree induction), which can then be concatenated into networks representing word pronunciations (Chen 1990; Riley 1991). A detailed comparison of the two approaches is desirable, but so far hasn't been carried out. We simply remark that both approaches could be combined by generating allophone

[17] It is customary in HMM modeling for speech recognition to introduce *self-loops* on states to model varying durations. This does not contradict what was said above; these self-loops are introduced at a lower representational level, along with other devices such as state replication. They are systematically added to the merged HMM before using the HMM for alignment with continuous speech data.

sequences from induced phoneme-based models and adding them to directly observed pronunciations for the purpose of smoothing.

3.6.2.3 Quantitative evaluation

Several HMM construction methods were tested and compared using the TIMIT data.

- The maximum-likelihood (ML) model: the HMM is the union of all unique samples, with probabilities corresponding to the observed relative frequencies. This is essentially the result of building the initial model in the HMM merging procedure, before any merging takes place.

- Baum-Welch estimation: an HMM of fixed size and structure is submitted to Baum-Welch (EM) estimation of the probability parameters. Of course, finding the 'right' size and structure is exactly the learning problem at hand. We wanted to evaluate the structure finding abilities of the Baum-Welch procedure, so we set the number of states to a fixed multiple of the maximum sample length for a given word, and randomly initialized all possible transitions and emissions to non-zero probabilities. After the EM procedure converges, transitions and emissions with probability close to zero are pruned, leaving an HMM structure that can be evaluated. Several model sizes (as multiples of the sample length) were tried.

- Standard HMM merging with loop suppression (see above). We used the simple description length prior from Section 3.3.3.3, with $\log P = -n_t \log |Q|$ for transitions and $\log P = -n_e \log |\Sigma|$ for emissions, as well as a narrow Dirichlet prior for transition and emission probabilities, with $\alpha_0 = 1.0$ in both cases. Several global weighting factors λ for the structure prior were evaluated.

- HMM merging with the single-output constraint, as explained above. The same prior as for the multiple-output HMMs was used, except that an emission parameter prior does not exist in this case, and the structural prior contribution for an emission is $\log P = -\log |\Sigma|$.

A simple-minded way of comparing these various methods would be to apply them to the training portion of the data, and then compare generalization on the test data. As a measure of generalization it is customary to use the negative log probability, or empirical cross-entropy, they assign to the test samples. The method that achieves the lowest cross-entropy would 'win' the comparison.

A problem that immediately poses itself is that there is a significant chance that some of the test samples have zero probability according to the induced HMMs. One might be tempted to evaluate based on the number of test samples covered by the model, but such a comparison alone would be meaningless since a model that assigns (very low) probability to all possible strings could trivially 'win' in this comparison.

The general approach that is usually taken in this situation is to have some recipe that prevents vanishing probabilities on new, unseen samples. There are a great many such approaches in common use, such as parameter smoothing and back-off schemes, but many of these are not suitable for the comparison task at hand.

At the very least, the method chosen should be

- well-defined, *i.e.*, correspond to some probabilistic model that represents a proper distribution over all strings;
- unbiased with respect to the methods being compared, to the extent possible.

Standard back-off models (where a second model is consulted if, and only if, the first one returns probability zero) do not yield consistent probabilities unless they are combined with 'discounting' of probabilities to ensure that the total probability mass sums up to unity (Katz 1987). The discounting scheme, as well as various smoothing approaches (*e.g.*, adding a fixed number of virtual 'Dirichlet' samples into parameter estimates) tend to be specific to the model used, and are therefore inherently problematic when comparing different model-building methods.

To overcome these problems, we chose to use the mixture models approach described in Section 2.3.1. The target models to be evaluated are combined with a simple back-off model that guarantees non-zero probabilities, *e.g.*, a bigram grammar with smoothed parameters. This back-off grammar is identical in structure for all target models. Unlike discrete back-off schemes, the target and the back-up are always consulted both for the probability they assign to a given sample, which are then weighted and averaged according to a mixture proportion.

When comparing two model induction methods, we first let each induce a structure. Each is built into a mixture model, and both the component model parameters and the mixture proportions are estimated using the EM procedure for generic mixture distributions. To get meaningful estimates for the mixture proportions, the HMM structure is induced based on a subset of the training data, and the full training data is then used to estimate the parameters, including the mixture weights. This holding-out of training data makes the mixture model approach similar to the deleted interpolation method (Jelinek & Mercer 1980). The main difference is that the component parameters are estimated jointly with the mixture proportions.[18] In our experiments we always used half of the training data in the structure induction phase, adding the other half during the EM estimation phase. Also, to ensure that the back-off model receives a non-zero prior probability, we estimate the mixture proportions under a simple symmetrical Dirichlet prior with $\alpha_1 = \alpha_2 = 1.5$.

3.6.2.4 Results and discussion

HMM merging was evaluated in two variants, with and without the single-output constraint. In each version, three settings of the structure prior weight λ were tried: 0.25, 0.5 and 1.0. Similarly, for Baum-Welch training the preset number of states in the fully parameterized HMM was set to 1.0, 1.5 and 1.75 times the longest sample length. For comparison purposes, we also included the performance of the unmerged maximum-likelihood HMM, and a biphone grammar of the kind used in the mixture models used to evaluate the other model types. Table 3.1 summarizes the results of these experiments.

[18] This difference can be traced to the different goals: in deleted interpolation the main goal is to gauge the reliability of parameter estimates, whereas here we want to assess the different structures.

	ML	M ($\lambda = 0.25$)	M ($\lambda = 0.5$)	M ($\lambda = 1.0$)
log P	$-2.600 \cdot 10^3$	$-2.418 \cdot 10^3$	$-2.355 \cdot 10^3$	$-2.343 \cdot 10^3$
Perplexity	1.979	1.886	1.855	**1.849**
Significance	$p < 0.000001$	$p < 0.0036$	$p < 0.45$	–
states	4084	1333	1232	1204
transitions	4857	1725	1579	1542
emissions	3878	1425	1385	1384
training time	28:19	32:03	28:58	29:49

	ML	M1 ($\lambda = 0.25$)	M1 ($\lambda = 0.5$)	M1 ($\lambda = 1.0$)
log P	$-2.600 \cdot 10^3$	$-2.450 \cdot 10^3$	$-2.403 \cdot 10^3$	$-2.394 \cdot 10^3$
Perplexity	1.979	1.902	1.879	1.874
Significance	$p < 0.000001$	$p < 0.0004$	$p < 0.013$	$p < 0.016$
states	4084	1653	1601	1592
transitions	4857	2368	2329	2333
emissions	3878	1447	1395	1386
training time	28:19	30:14	26:03	25:53

	BG	BW ($N = 1.0L$)	BW ($N = 1.5L$)	BW ($N = 1.75L$)
log P	$-2.613 \cdot 10^3$	$-2.470 \cdot 10^3$	$-2.385 \cdot 10^3$	$-2.392 \cdot 10^3$
Perplexity	1.985	1.912	*1.870*	1.873
Significance	$p < 0.000001$	$p < 0.000003$	$p < 0.041$	$p < 0.017$
states	n/a	1120	1578	1798
transitions	n/a	1532	2585	3272
emissions	n/a	1488	1960	2209
training time	3:47	55:36	99:55	123:59

Table 3.1: Results of TIMIT trials with several model building methods.
The training methods are identified by the following keys: BG bigram grammar, ML maximum likelihood HMM, BW Baum-Welch trained HMM, M merged HMM, M1 single-output merged HMM. log P is the total log probability on the 1895 test samples. Perplexity is the average number of phones that can follow in any given context within a word (computed as the exponential of the per-phone cross-entropy). Significance refers to the p level in a t-test pairing the log probabilities of the test samples with those of the best score (merging, $\lambda = 1.0$).

The number of states, transitions and emissions is listed for the resulting HMMs where applicable. The training times listed represent the total time (in minutes and seconds) it took to induce the HMM structure and subsequently EM-train the mixture models, on a SPARCstation 10/41.

The table also includes useful summary statistics of the model sizes obtained, and the time it took to compute the models. The latter figures are obviously only a very rough measure of computational demands, and their comparison suffers from the fact that the implementation of each of the methods may certainly be optimized in idiosyncratic ways. Nevertheless these figures should give an approximate idea of what to expect in a realistic application of the induction methods involved.

One important general conclusion from these experiments is that both the merged models and those obtained by Baum-Welch training do significantly better than the two 'dumb' approaches, the bigram grammar and the ML HMM (which is essentially a list of observed samples). We can therefore conclude that it pays to try to generalize from the data, either using our Bayesian approach or Baum-Welch on an HMM of suitable size.

Overall the difference in scores even between the simplest approach (bigram) and the best scoring one (merging, $\lambda = 1.0$) are quite small, with phone perplexities ranging from 1.985 to 1.849. This is not surprising given the specialized nature and small size of the sample corpus. Unfortunately, this also leaves very little room for significant differences in comparing alternate methods. However, the advantage of the best model merging result (unconstrained outputs with $\lambda = 1.0$) is still significant compared to the best Baum-Welch (size factor 1.5) result ($p < 0.041$). Such small differences in log probabilities would probably be irrelevant when the resulting HMMs are embedded in a speech recognition system.

Perhaps the biggest advantage of the merging approach in this application is the compactness of the resulting models. The merged models are considerably smaller than the comparable Baum-Welch HMMs. This is important for any of the standard algorithms operating on HMMs, which typically scale linearly with the number of transitions (or quadratically with the number of states). Besides this advantage in production use, the training times for Baum-Welch grow quadratically with the number of states for the structure induction phase since it requires fully parameterized HMMs. This scaling is clearly visible in the run times we observed.

Although we haven't done a word-by-word comparison of the HMM structures derived by merging and Baum-Welch, the summary of model sizes seem to confirm our earlier finding (Section 3.6.1) that Baum-Welch training needs a certain redundancy in 'model real estate' to be effective in finding good-fitting models. Smaller size factors give poor fits, whereas sufficiently large HMMs will tend to overfit the training data.

The choice of the prior weights λ for HMM merging (Section 3.4.4) controls the model size in an indirect way: larger values lead to more generalization and smaller HMMs. For best results this value can be set based on previous experience with representative data. This could effectively be done in a cross-validation like procedure, in which generalization is successively increased starting with small λ's. Due to the nature of the merging algorithm, this can be done incrementally, *i.e.*, the outcome of merging with a small λ can be submitted to more merging at a larger λ value, until further increases reduce generalization on the cross-validation data.

3.6.3 Multiple pronunciation word models for speech recognition

As part of his dissertation research, Wooters (1993) has used HMM merging extensively in the context of the Berkeley Restaurant Project (BeRP). BeRP is medium vocabulary, speaker-independent spontaneous continuous speech understanding system that functions as a consultant for finding restaurants in the city of Berkeley, California (Jurafsky *et al.* 1994a).

In this application, the merging algorithm is run on strings of phone labels obtained by Viterbi-aligning previously existing word models to sample speech (using the TIMIT labels as the phone alphabet). As a result, new word models are obtained, which are then again used for Viterbi alignment, leading to improved labelings, *etc.* This procedure is iterated until no further improvement in the recognition performance (or the labelings themselves) is observed. The word models are bootstrapped with a list of pronunciations from a variety of databases. The goal of the iteration with repeated alignments and mergings is to tailor the word models to the task-specific data at hand, and to generalize from it where possible.

An added complication is that HMMs with discrete outputs are not by themselves applicable to acoustic speech data. Using an approach developed by Bourlard & Morgan (1993), the HMMs are combined with acoustic feature densities represented by a multi-layer perceptron (MLP). This neural network maps each frame of acoustic features into the phone alphabet. From the network outputs, the likelihoods of the HMM states relative to the acoustic emissions can be computed, as required for the Viterbi alignment or other standard HMM algorithms.

Since these emission probabilities are also subject to change due to changes of the word models, they too have to be reestimated on each iteration. The MLP is bootstrapped with weights obtained by training on the pre-labeled TIMIT acoustic data. Figure 3.15 depicts the combined iteration of MLP training, word model merging, and Viterbi alignment. It can be viewed as an instance of a generalized EM algorithm, in which emission probabilities (represented by the MLP) and HMM structure and transition probabilities are optimized separately. The separation is a result of the different technologies used.

For the BeRP system, HMM merging made it possible and practical to use multiple pronunciation word models, whereas before it was confined to single pronunciation models. (Note that in this setting, even a very restricted HMM can produce any acoustic emission with non-zero probability, due to the continuous nature of the domain, and because the emission distribution represented by the MLP is inherently non-vanishing.)

To assess its effectiveness, the recognition performance of the multiple-pronunciation system was compared against that of an otherwise identical system in which only one phone sequence per word was used (generated by a commercial text-to-speech system). In this comparison, multiple-pronunciation modeling as derived by HMM merging was found to reduce the word-level error rate from 40.6% to 32.1%. At the same time, the error rate at the level of semantic interpretations dropped from 43.4% to 34.1%.

Further experiments are needed to identify more precisely what aspects of the multiple-pronunciation approach account for the improvement, *i.e.*, whether other word model building techniques would lead to significantly different results in this context. However, the preliminary results do show that HMM merging is

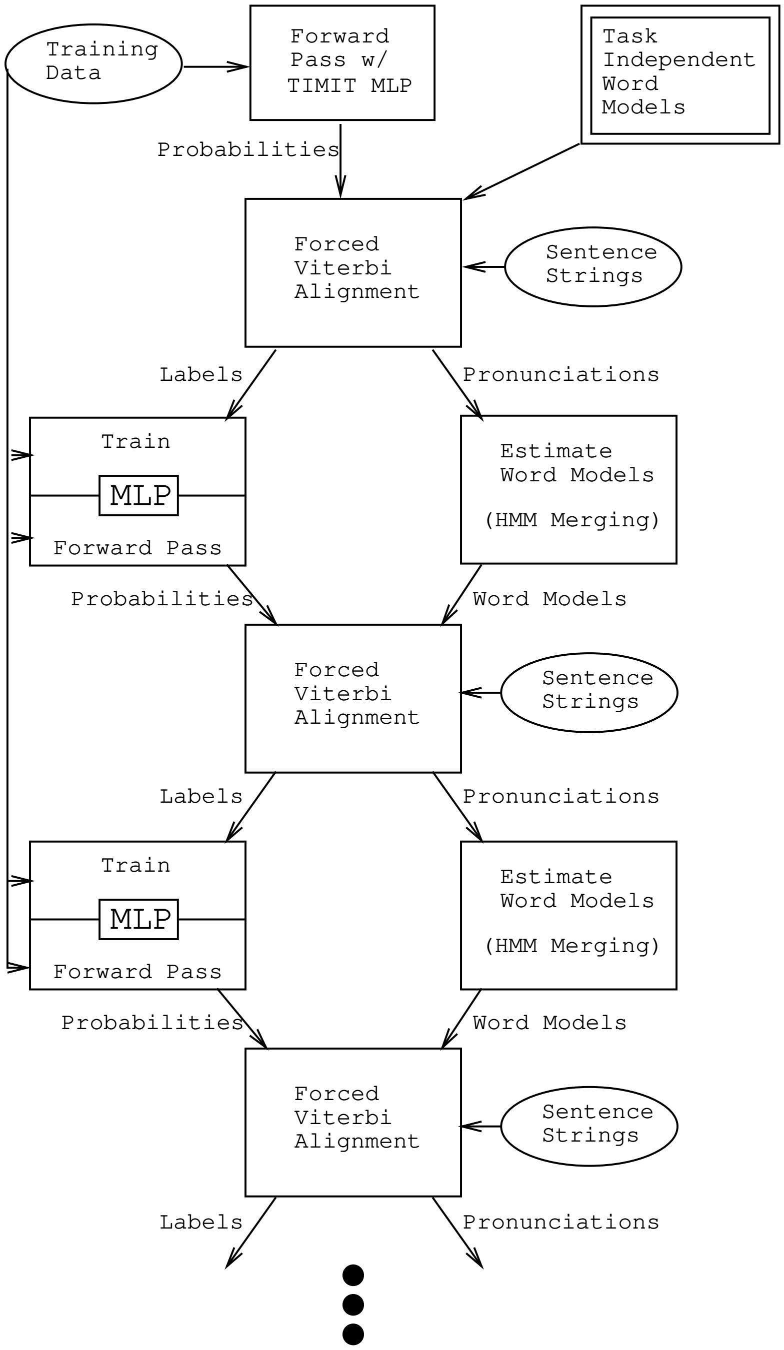


Figure 3.15: Hybrid MLP/HMM training/merging procedure used in the BeRP speech understanding system (Wooters 1993).

both practical and effective when embedded in a realistic speech system.

The details of the construction of these word models, along with discussion of ancillary issues and a graphical HMM representation of the pronunciation for the 50 most common words in the BeRP corpus can be found in Wooters (1993).

3.7 Conclusions and Further Research

Our evaluations indicate that the HMM merging approach is a promising new way to induce probabilistic finite-state models from data. It compares favorably with the standard Baum-Welch method, especially when there are few prior constraints on the HMM topology. Our implementation of the algorithm and applications in the speech domain have shown it to be feasible in practice.

Experimentation with the range of plausible priors, as well as new, application-specific ones is time-consuming, and we have barely scratched the surface in this area. However, the experience so far with the priors discussed in Section 3.3.3 is that the particular choice of prior type and parameters does not greatly affect the course of the best-first search, except possibly the decision when to stop. In other words, the merging heuristic, together with the effect of the likelihood are the determining factors in the choice of merges. This could, and in fact should, change with the use of more informative priors.

Likewise, we haven't pursued merging of HMMs with non-discrete outputs. For example, HMMs with mixtures of Gaussians as emission densities are being used extensively (Gauvain & Lee 1991) for speech modeling. Our merging algorithm becomes applicable to such models provided that one has a prior for such densities, which should be straightforward (Cheeseman *et al.* 1988). Efficient implementation of the merging operator may be a bigger problem—one wants to avoid having to explicitly compute a merged density for each merge under consideration.

One of the major shortcomings of the current merging strategy is its inability to 'back off' from a merging step that turns out be an overgeneralization in the light of new data. A solution to this problem might be the addition of a complementary state *splitting* operator, along the lines of Bell *et al.* (1990) or Ron *et al.* (1994). The evaluation functions used in those approaches are entropy-based, and thus equivalent to maximum likelihood, which also means that a Bayesian generalization (by adding a suitable prior) should not be hard.

As mentioned in Section 3.5.3, standard splitting alone will not explore the full power of finite-state models, and combining merging with splitting would circumvent this limitation. The major difficulty with evaluating splits (as opposed to merges) is that it requires rather more elaborate statistics than simple Viterbi counts, since splitting decisions are based on co-occurrences of states in a path.

Chapter 4

Stochastic Context-free Grammars

4.1 Introduction and Overview

In this chapter we will look at model merging as applied to the probabilistic version of context-free grammars. The stochastic context-free grammar (SCFG) formalism is a generalization of the HMM, just as non-probabilistic CFGs can be thought of as an extension of finite state grammars.

Unlike their their non-probabilistic counterpart, SCFGs are not a 'mainstream' approach to language modeling yet.[1] In most of today's probabilistic language models finite-state or even simple n-gram approaches dominate. One reason for this is that although most standard algorithms for probabilistic finite-state models (i.e., HMMs) have generalized versions for SCFGs, they become computationally more demanding, and often intractable in practice (see Section 4.2.2).

A more important problem is that SCFGs may actually be worse at modeling one aspect of language in which simple finite-state models do a surprisingly good job: capturing the short-distance, lexical (as opposed to phrase-structural) contingencies between words. This is a direct consequence of the conditional independence assumptions embodied in SCFGs, and has prompted the investigation of 'mildly context-sensitive' grammars and their probabilistic versions (Resnik 1992; Schabes 1992). These, however, come at an even greater computational price.

Recent work has shown that probabilistic CFGs can be useful if applied carefully and in the right domain. Lari & Young (1991) discuss various applications of estimated SCFGs for phonetic modeling. Jurafsky *et al.* (1994b) show that a SCFG built from hand-crafted rules with probabilities estimated from a corpus can improve speech recognition performance over standard n-gram language models, either by directly coupling the SCFG to the speech decoder, or by using the SCFG effectively as a smoothing device to improve the estimates of n-gram probabilities from sparse data. The algorithms that form the basis of these last two approaches are described in the second part of this thesis, in Chapter 6 and Chapter 7, respectively.

[1] While bare CFGs aren't widely used in computational linguistics either, they form the basis or 'backbone' of most of today's feature and unification-based grammar formalisms, such as LFG (Kaplan & Bresnan 1982), GPSG (Gazdar *et al.* 1985), and construction grammar (Fillmore 1988).

Finally, SCFGs have been applied successfully to the modeling of biological structures, notably the secondary structures of various types of RNA (Sakakibara *et al.* 1994; Underwood 1994).[2] It seems that biological sequences with their relatively small alphabets and more precisely formalized combinatorial properties are comparatively ideal applications for formal grammars, next to the rather messy natural language domain.[3]

In short, we will leave open the question of whether context-free models are appropriate or useful in general, and focus instead on the learning problem itself, within the Bayesian model merging framework developed earlier.

The following Section 4.2 reviews the basic SCFG formalism and gives an overview of the standard algorithms as they relate to the learning problem.

Following that, Section 4.3 describes the model merging algorithm for SCFGs. It is not surprising that many of the concepts and techniques introduced in Chapter 3 reappear here in similar form. The major difference is the introduction of an additional merging operator, called *chunking* or *syntagmatic merging*.

Section 4.4 relates the Bayesian SCFG merging method to various other, probabilistic and non-probabilistic approaches found in the literature.

In Section 4.5 we evaluate the algorithm experimentally using various formal and quasi-natural language target grammars. Section 4.5.4 summarizes and discusses possible extensions and approaches to problems in the current method.

4.2 Stochastic Context-free Grammars

4.2.1 Definitions

Definition 4.1 A *stochastic context-free grammar (SCFG)* M consists of

a) a set of *nonterminal symbols* $\mathcal{N}$,

b) a set of *terminal symbols* (or *alphabet*) Σ,

c) a *start nonterminal* $S \in \mathcal{N}$,

d) a set of *productions* or *rules* $\mathcal{R}$,

e) production probabilities $P(r)$ for all $r \in \mathcal{R}$.

The productions are of the form

$$X \rightarrow \lambda \quad ,$$

where $X \in \mathcal{N}$ and $\lambda \in (\mathcal{N} \cup \Sigma)^*$. X is called the *left-hand side (LHS)* of the production, whereas λ is the *right-hand side (RHS)*.

[2] Work in progress addresses the structure of proteins (Kevin Thompson, personal communication).

[3] This constitutes a subjective judgment from a computational linguist's point-of-view. Many molecular biologists would probably disagree and claim that language is the comparatively simpler domain.

Notation Latin capital letters X, Y, Z denote nonterminal symbols. Latin lowercase letters $a, b, \ldots$ are used for terminal symbols. Strings of mixed nonterminal and terminal symbols are written using lowercase Greek letters λ, μ, ν. The empty string is denoted by ϵ.

A SCFG is thus exactly like a standard context-free grammar, except that productions are assigned continuous probability parameters. The interpretation of $P(X \rightarrow \lambda)$ is that in a top-down derivation from the SCFG, the RHS λ is chosen with the indicated probability whenever nonterminal X is to be expanded. $P(X \rightarrow \lambda)$ is thus a probability conditioned on X. Due to this interpretation we get the well-formedness condition

$$\sum_{\lambda} P(X \rightarrow \lambda) = 1 \quad ,$$

where the sum is over all productions with LHS X.

Unfortunately, this simple local consistency of the probability parameters in a SCFG is not sufficient to necessarily let the SCFG represent a proper distribution over strings. This complication does not come into play during learning of SCFGs from corpora, but it is important when dealing with predetermined SCFGs for parsing and other purposes. We therefore defer discussion of this technicality to later chapters (Sections 6.4.8 and 7.5).

As for HMMs and other probabilistic grammars, the conditional probability of a string given a SCFG M is the sum of the probabilities of all derivations of the string. This can be formalized as follows.

Definition 4.2 a) A *sentential form* of M is a string ν of nonterminals and terminals, such that either $\nu = S$, or there is a sentential form μ from which ν can be produced by replacing one nonterminal according to a production of M, i.e., $\mu = \mu_1 X \mu_2$, $\nu = \mu_1 \lambda \mu_2$, and $X \rightarrow \lambda \in \mathcal{R}$.

b) A *(left-most) derivation* in M is a sequence of sentential forms beginning with S, each derived by a single rule application from its predecessor, such that at each step the replaced nonterminal X is always the left-most nonterminal in the sentential form, i.e., $\mu_1 \in \Sigma^*$ in the notation above. We write a derivation as $S \Rightarrow \nu_2 \Rightarrow \ldots \nu_k$, where $\nu_2, \ldots, \nu_k$ are the sentential forms involved.

c) The *probability of a derivation* $S \Rightarrow \nu_2 \Rightarrow \ldots \nu_k$ is inductively defined by

1) $P(S) = 1$

2) $P(S \Rightarrow \ldots \Rightarrow \nu_k) = P(S \Rightarrow \ldots \Rightarrow \nu_{k-1}) P(X \rightarrow \lambda)$,

where $X \rightarrow \lambda$ is the production used in the step $\nu_{k-1} \Rightarrow \nu_k$.

d) The *probability of a string* x in M is

$$P(x|M) = \sum_{S \Rightarrow \ldots \Rightarrow x} P(S \Rightarrow \ldots \Rightarrow x) \quad , \tag{4.1}$$

where the summation is over all derivations that end in ((or *yield*) x.

The definition of derivation and string probabilities is thus completely analogous to HMMs, although the notion of derivation itself is somewhat more complex due to the branching character of a CFG. Left-most derivations are isomorphic to parse trees in the usual way, and the probability of a derivation is just the product of all the rule probabilities involved in the generation. The restriction to left-most derivations avoids counting a derivation multiple times when computing the total string probability. As usual, we say a string is ambiguous if it has more than one derivation (with non-zero probability).

A string may have probability zero either because it cannot be derived due to missing productions, or because all its derivations have zero probability (because at least one rule has probability zero in each of them). For computing string probabilities, it makes no difference whether a rule is 'not there' or whether it is present but has zero probability. The difference becomes relevant, however, when considering possible parameterizations of SCFGs and corresponding prior distributions (Section 4.3.4).

4.2.2 SCFG estimation

In estimating the rule probabilities of an SCFG, the problem of the unobservable derivations has to be overcome, just as for HMMs and other probabilistic grammars with hidden derivation processes. The standard solution is again based on the EM algorithm (see Section 2.3.2).

The sufficient statistics for estimating the rule probabilities are the counts of how many times each rule is used in the derivations generating a corpus, denoted by $c(X \to \lambda)$ for production $X \to \lambda$. The maximum-likelihood (ML) estimates are obtained by instantiating equation 2.8,

$$\hat{P}(X \to \lambda) = \frac{c(X \to \lambda)}{\sum_{\mu} c(X \to \mu)} \quad .$$

For the EM procedure to go through, the *expected counts* $\hat{c}(X \to \lambda)$, given the string samples and a set of current rule probabilities have to be computed (the E-step). Following that, the rules are reestimated using $\hat{c}$ instead of c in the above formula (the M-step), and the two steps are iterated to converge to a local likelihood maximum.

In the presence of parameter priors the estimates can be modified accordingly. For example, with a Dirichlet prior for each nonterminal X, a maximum posterior (MAP) estimate is obtained by applying equation (2.17) to the rule counts instead. When using the MAP estimates, the EM procedure finds parameters that locally maximize the posterior probabilities, instead of just the likelihood.

As for HMMs, dynamic programming (in the guise of *chart parsing*) can be used to compute the expected counts $\hat{c}$ with relative efficiency. The dynamic programming scheme for SCFGs is known as the Inside-Outside algorithm (Baker 1979). It was originally formulated for SCFGs in Chomsky Normal Form (CNF), but has since been generalized to cope with arbitrary SCFGs. Rule estimation is also part of the probabilistic Earley parsing framework described in Chapter 6 (Section 6.5.2). The complexity of the Inside-Outside computation is $O(\ell^3 |\mathcal{N}|^3)$, where ℓ is the length of the input. This makes the estimation algorithm quite a bit more expensive than the forward-backward algorithm used for HMMs. Complexity and ways to reduce it are also discussed in Chapter 6.

4.2.3 Viterbi parses

The most likely derivation for a given string is the *Viterbi parse*. As in the case of HMMs we will simplify the probability computations involved in the model merging algorithm by keeping track only of the statistics generated by Viterbi parses.

Viterbi parses are computed by dynamic programming. A chart is filled bottom-up, computing for each substring of a sentence and each nonterminal the partial parse with highest probability. Once the chart is completed, the maximum probability parse can be traced back from the root entry. A version of this method based on Earley-style chart parsing is described in Chapter 6.

4.3 SCFG Merging

We now describe the ingredients for extending the model merging approach to SCFGs. As before, four issues have to be addressed (cf. Section 3.3):

1. How to incorporate samples into the initial or current model?
2. What are the merging operators?
3. How to evaluate the quality of a model?
4. How to search the model space, based on the chosen operators and evaluation function?

We start by presenting the sample incorporation and merging operators, without regard to the evaluation function, then walk through an example to provide some intuition for the mechanics of the process. Priors for SCFGs are discussed next which, combined with the standard likelihood function for SCFGs (equation 4.1), give us a posterior probability to evaluate models. The SCFG merging algorithm is then completed by specifying the search strategies in the maximization of the posterior probability over the model space.

4.3.1 Sample incorporation and merging operators

4.3.1.1 Initial rules

The goal of sample incorporation in model merging is to produce an initial (or incremental) model that accounts for all samples observed so far, typically by dedicating simple submodels to each new sample. For SCFGs this can be accomplished with *dedicated top-level productions*, each designed to generate one sample.

Specifically, to incorporate a sample $a_1 a_2 \ldots a_\ell$, we create new nonterminals $X_1, X_2, \ldots, X_\ell$ and add the following productions to the current grammar:

$$S \quad \rightarrow \quad X_1 X_2 \ldots X_\ell$$

$$\begin{aligned}X_1 &\rightarrow a_1\\ X_2 &\rightarrow a_2\\ &\vdots\\ X_\ell &\rightarrow a_\ell\end{aligned}$$

The reason for not simply adding a single production

$$S \rightarrow a_1 a_2 \ldots a_\ell$$

is to keep grammar in a special format that simplifies the definition of the merging operators. The initial rules, and all those created thereafter by merging, will fall into one of two categories:

- *Nonterminal productions* with any (non-zero) number of nonterminals on the RHS.
- *Terminal* or *lexical productions* with a single terminal on the RHS. The LHS nonterminals in these productions are also called *preterminals*.

To refer to this type of CFG, and for lack of a better term, we call this of type of rule format *Simple Normal Form (SNF)*. Any general CFG without null productions can be converted to SNF by possibly 'shadowing' some terminals with preterminals, while keeping the structure of derivations intact. SNF is thus a weaker constraint than Chomsky Normal Form (CNF), which imposes a binary branching structure on derivations.

4.3.1.2 Keeping count

Along with manipulating the grammar productions, we will keep track of usage counts for each rule in the grammar. These statistics are based on the Viterbi parses for the samples from which the grammar was built.

In the Bayesian setting, counts are more informative than mere probabilities. As we have seen in Section 3.4.3 in the case of HMMs, Viterbi counts can be used to estimate probabilities and/or compute approximate posterior probabilities for the grammar structure, effectively integrating out rule probabilities and treating them as 'nuisance parameters.'

As before, Viterbi counts can be obtained efficiently without parsing (and re-parsing) each sample by updating rule counts on each merging operation under the assumption that the Viterbi parses are preserved by the merging operators. As part of this strategy, we also collapse multiple identical samples during the incorporation step and assign corresponding counts to the initial productions. These will be notated in parentheses next to the productions.

Thus, a c-fold occurrence of sample $a_1 a_2 \ldots a_\ell$ would create the following rules and associated counts:

$$\begin{aligned}S &\rightarrow X_1 X_2 \ldots X_\ell \quad (c)\\ X_1 &\rightarrow a_1 \quad (c)\end{aligned}$$

$$
\begin{aligned}
X_2 &\rightarrow a_2 \quad (c) \\
&\vdots \\
X_\ell &\rightarrow a_\ell \quad (c)
\end{aligned}
$$

Notation Counts and probabilities will be distinguished by parentheses and brackets, respectively. Thus

$$S \rightarrow X_1 X_2 \dots X_\ell \quad [p]$$

indicates the rule probability $p = P(S \rightarrow X_1 X_2 \dots X_\ell)$. The same distinction is applied when listing sets of sample strings.

$$\texttt{abc} \quad (4)$$

denotes a 4-fold occurrence of string `abc`, whereas

$$\texttt{abc} \quad [0.12]$$

means that `abc` has probability 0.12 relative to some distribution.

4.3.1.3 Nonterminal merging

When rewriting a finite-state grammar as a CFG, the states turn into nonterminals in left or right-linear productions. It is thus natural to generalize the state merging operation used in HMMs to nonterminals in SCFGs.

The nonterminal merging operator replaces two existing nonterminals X_1 and X_2 with a single new nonterminal Y. We will use merge$(X_1, X_2) = Y$ as a shorthand for the operation. It has a twofold effect on grammars:

1. RHS occurrences of X_1 and X_2 are replaced by Y:

$$
\begin{aligned}
Z_1 &\rightarrow \lambda_1 X_1 \mu_1 \quad (c_1) \\
Z_1 &\rightarrow \lambda_2 X_2 \mu_2 \quad (c_2) \\
&\Downarrow \ \text{merge}(X_1, X_2) = Y \\
Z_1 &\rightarrow \lambda_1 Y \mu_1 \quad (c_1) \\
Z_2 &\rightarrow \lambda_2 Y \mu_2 \quad (c_2)
\end{aligned}
$$

   Note that this may lead to the merging of entire productions, namely, if $Z_1 = Z_2$, $\lambda_1 = \lambda_2$, and $\mu_1 = \mu_2$. In this case the count of the merged production would be $c_1 + c_2$.

2. The union of the productions for the LHSs X_1 and X_2 is associated with the new nonterminal Y:

$$
\begin{aligned}
X_1 &\rightarrow \lambda_1 \quad (c_1) \\
X_2 &\rightarrow \lambda_2 \quad (c_2)
\end{aligned}
$$

$$\begin{array}{rcl} & \Big\Downarrow & \text{merge}(X_1, X_2) = Y \\ Y & \rightarrow & \lambda_1 \quad (c_1) \\ & \rightarrow & \lambda_2 \quad (c_2) \end{array}$$

Here we assume that λ_1 and λ_2 have previously been subjected to substitution as per step 1. LHS merging can also lead to entire productions becoming merged, if $\lambda_1 = \lambda_2$. The new count would then be $c_1 + c_2$.

The analogy to HMM state merging carries further. In HMMs, a merging operation may generate a loop, thereby implementing the 'inductive leap' from a model that generates only a finite number of strings to one that allows for infinitely many strings. The same is true here: merging can create recursion in the productions, such as when X_1 and X_2 occur in the LHS and RHS, respectively, of the same production:

$$\begin{array}{rcl} X_1 & \rightarrow & \lambda X_2 \mu \\ & \Big\Downarrow & \text{merge}(X_1, X_2) = Y \\ Y & \rightarrow & \lambda Y \mu \end{array}$$

Merging may also introduce recursion through intermediate nonterminals.[4]

There is one special case of recursion that has to be dealt with specially. Whenever nonterminal merging creates a rule

$$Y \rightarrow Y \quad ,$$

it is deleted from the grammar. The rationale here is that such rules do not contribute anything to derivations, as chains of Y's can always be replaced by a single Y without otherwise affecting the structure of the derivation. (This also implies that the counts associated with such rules can safely be 'forgotten.')

The convenience of the SNF format stems from the fact that nonterminal merging can be applied directly to the initial grammar. Otherwise preterminals would have to be created specifically for this purpose, a gratuitous complication of the merging operator.

4.3.1.4 Nonterminal chunking

Nonterminal merging alone cannot create context-free productions with the usual embedding structure; therefore an additional operator is needed. A natural extension that (combined with merging) will enable all possible CFG structures to be generated is the following: given an *ordered sequence* of nonterminals $X_1 X_2 \ldots X_k$, create a new nonterminal Y that expands to $X_1 X_2 \ldots X_k$, and replace occurrences of $X_1 X_2 \ldots X_k$ in RHSs with Y. We will call this operation *chunking* and write it as chunk$(X_1 X_2 \ldots X_k) = Y$.

$$\begin{array}{rcl} Z & \rightarrow & \lambda X_1 X_2 \ldots X_k \mu \quad (c) \\ & \Big\Downarrow & \text{chunk}(X_1 X_2 \ldots X_k) = Y \end{array}$$

[4] The simple direct recursion for SCFGs would roughly correspond to a self-loop on an HMM state. In general loops will involve several intermediate states, of course.

$$\begin{aligned} Z &\rightarrow \lambda Y \mu \quad (c) \\ Y &\rightarrow X_1 X_2 \ldots X_k \quad (c') \end{aligned}$$

The count c' of the newly created production is the sum of the counts all productions where substitutions with Y have occurred.

Chunking can be viewed as a different kind of merging operation. Instead of identifying two *alternative* submodels (sublanguages generated by nonterminals) as being the 'same,' chunking hypothesizes that two submodels form a unit by virtue of *co-occurrence*.[5] These two ways of creating new units of description correspond exactly to the structuralist distinction between paradigmatic and syntagmatic relations among linguistic elements (de Saussure 1916; Hjelmslev 1953). A more explicit terminology would therefore use *paradigmatic merging* for the standard nonterminal merging operation, and *syntagmatic merging* for the chunking operation. However, we will keep the short names 'merging' and 'chunking' both for brevity and to highlight the similarity of HMM state and SCFG nonterminal merging.

Chunking by itself does not enable a grammar to generate new strings, and in SCFGs all strings retain their probability as a result of the operation. Therefore, unlike merging, an isolated chunking step involves no generalization beyond the strings already generated. However, the nonterminal created by chunking may itself feed into subsequent merging steps, thereby producing generalizations that wouldn't otherwise have been possible—this is precisely the purpose of introducing a second operator.

In principle chunking and merging can create any CFG structure in SNF, subject to the search involved in finding the right sequence of operations, and assuming enough samples are available. This can be seen as follows: given a target grammar, we need a sample corpus that covers every rule in the grammar, i.e., such that each production was used at least once to generate the corpus. After building the initial grammar of flat productions as described in Section 4.3.1.1, we can then recreate the target grammar by merging and chunking from the bottom up so as to obtain the rules in the generating grammar. Specifically, compare the derivations of the incorporated samples to the derivations in the target grammar that generated them, merging exactly those nonterminals that map to the same target nonterminal, and chunking exactly those nonterminals that form phrasal units in the target grammar.

4.3.2 An example

A simple formal language example will illustrate the interaction of sample incorporation and merging operators. The target grammar generates the language $\{a^n b^n, n > 0\}$, and can be written in SNF as

```
S  --> A B
   --> A S B
A  --> a
B  --> b
```

Suppose the samples and their counts are

[5] Actually, there is an implicit restriction on *contiguous co-occurrence*, which is part of the CFG framework. As a result, discontinuous syntactic units cannot be directly represented.

```
      a b          (10)
    a a b b        (5)
  a a a b b b      (2)
a a a a b b b b    (1)
```

The actual frequencies of occurrence of these samples are important in determining the result of learning, as the likelihood function of the grammar depends on them. We list the actual log likelihood values (using ML parameter estimates) after each step, so as to give an idea of the 'data fit' component of the eventual evaluation function.

For the initial grammar we adopt the convention that preterminal names are formed by appending a number to the uppercase version of the corresponding terminal symbol. `X`, `Y`, . . . denote nonterminals created otherwise, and `S` is reserved for the start nonterminal. This gives the following initial grammar:

```
S --> A1 B1                               (10)
  --> A2 A3 B2 B3                         (5)
  --> A4 A6 A7 B4 B5 B6                   (2)
  --> A8 A9 A10 A11 B7 B8 B9 B10 B11      (1)
A1 --> a                                  (10)
...                                       ...
A10 --> a                                 (1)
B1 --> b                                  (10)
...                                       ...
B10 --> b                                 (1)
```

(log likelihood = -8.50).

The next several merging steps will collapse the preterminals expanding to the same terminal, but leave the likelihood unchanged:

```
S --> A B                                 (10)
  --> A A B B                             (5)
  --> A A A B B B                         (2)
  --> A A A A B B B B B                   (1)
A --> a                                   (30)
B --> b                                   (30)
```

The first chunking operation, chunk(`A A B B`) = `X`, also preserves the likelihood, but shortens the existing rules. The shorter description will generally result in a higher prior probability for the grammar.

```
S --> A B                                 (10)
  --> X                                   (5)
  --> A X B                               (2)
  --> A A X B B                           (1)
X --> A A B B                             (8)
A --> a                                   (30)
B --> b                                   (30)
```

The new nonterminal `X` can now be merged with `S`:

```
S --> A B                       (10)
  --> A A B B                   (8)
  --> A S B                     (2)
  --> A A S B B                 (1)
A --> a                         (30)
B --> b                         (30)
```

(log likelihood = -8.71).

Another chunking operation, chunk(`A S B`) = `Y`, produces

```
S --> A B                       (10)
  --> A A B B                   (8)
  --> Y                         (2)
  --> A Y B                     (1)
Y --> A S B                     (3)
A --> a                         (30)
B --> b                         (30)
```

and again the new nonterminal `Y` is immediately merged with `S`:

```
S --> A B                       (10)
  --> A A B B                   (8)
  --> A S B                     (4)
A --> a                         (30)
B --> b                         (30)
```

(log likelihood = -8.69).

After one more chunking, chunk(`A B`) = `Z`, and subsequent merging step, merge(`Z`, `S`) = `S`, the grammar has its final form

```
S --> A B                       (18)
  --> A S B                     (12)
A --> a                         (30)
B --> b                         (30)
```

(log likelihood = -8.77).

As in the HMM example, it is instructive to consider additional merging steps, which would lead to various overly general grammars, such as

merge(`A`, `B`)	log likelihood = -26.83
merge(`S`, `A`)	log likelihood = -23.77
merge(`S`, `B`)	log likelihood = -23.77

This confirms our intuition that the right time to stop merging is characterized by a large drop in the likelihood. We should therefore see a posterior probability maximum at this point under any reasonable prior.

4.3.3 Bracketed samples

The major source of added difficulty in learning SCFGs, as opposed to finite-state models comes from the added uncertainty about the phrase structure (bracketing) of the observed samples. The chunking operation was created precisely to account for this part of the learning process.

Sample strings can be enriched to convey some of the phrase structure information. *Bracketed samples* are ordinary samples with substrings enclosed by balanced, non-overlapping pairs of parentheses. The bracketing need not be complete. For example, a partially bracketed sample for the example language of Section 4.3.2 is

```
(a a (a b) b b)
```

It is known that access to completely bracketed samples (equivalent to unlabeled derivation trees) makes learning non-probabilistic CFGs possible and tractable, by applying techniques borrowed from finite-state model induction (Sakakibara 1990). Pereira & Schabes (1992) have shown that providing even partial bracketing information can help the induction of properly structured SCFGs using the standard estimation approach. This raises the question how bracketed samples can be incorporated into the merging algorithm described so far.

This is indeed possible by a simple extension of the sample incorporation procedure described above. Instead of creating a single top-level production to account for a new sample, the algorithm creates a collection of productions and nonterminals to mirror the bracketing observed. Thus the sample `(a a (a b) b b)` is added to grammar using the productions

```
S --> A1 A2 X B2 B3                    (1)
X --> A3 B1                            (1)
```

plus lexical productions for the preterminals created. Each pair or parentheses generates an intermediate nonterminal, such as `X` above.

Merging and chunking are then applied to the resulting grammar as before. If the provided sample bracketing is *complete*, i.e., contains brackets for all phrase boundaries in the target grammar, then chunking becomes unnecessary. Merging alone can in principle produce the target grammar in this case, provided samples for all productions are given.

4.3.4 SCFG priors

In choosing prior distributions for SCFGs we again extend various approaches previously used for HMMs. As before, a model M is decomposed into a discrete structure M_S and collection of continuous parameters θ_M. Again, we have a choice of narrow or broad parameter priors, depending on whether the identity of non-zero rule probabilities is part of M_S or θ_M (Section 3.3.3). However, notice that broad parameter priors become problematic here since the set of possible rules (and hence parameters) is not a priori limited as for HMMs. As the length of RHSs grows, the number of potential rules over a given set of nonterminals also grows (exponentially). This makes narrow parameter priors inherently simpler and more natural for SCFGs.

The following combination of priors was used in the experiments reported below. The parameter prior $P(\theta_M|M_S)$ is a product of Dirichlet distributions (Section 2.5.5.1), one for each nonterminal. Each Dirichlet thus allocates prior probability over all possible expansions of a single nonterminal. The total prior

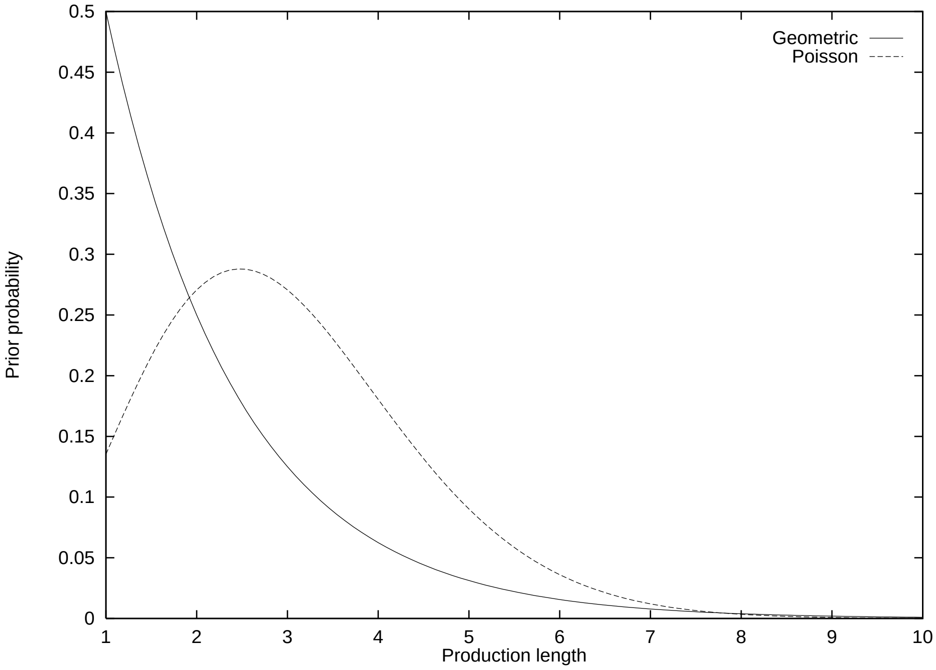


Figure 4.1: Geometric and Poisson length distributions, for identical distribution mean (3.0).

weights for each of the Dirichlets are equal and constant ($\alpha_0 = 1.0$). The Dirichlets are symmetrical, i.e., all expansions of a nonterminal are *a priori* equally likely. Again, we make the convenient simplification that the nonterminals are *a priori* independent in their expansions.

As for $P(M_S)$, two simple description length prior were used. In the first variant, each nonterminal production is encoded as a string of nonterminals and an end marker, each symbol requiring $\log |\mathcal{N}| + 1$ bits. Lexical productions need $\log |\Sigma|$ to encode.

While simple, this prior has the effect of letting the prior length of productions have the form of a geometric distribution, which is very unnatural, especially for natural language grammars. A second version of the description length prior was therefore devised which corresponds to production lengths drawn from a Poisson distribution

$$p(n;\mu) = \frac{e^{-\mu}\mu^n}{n!} \quad ,$$

where n is the length and μ its prior mean. Figure 4.1 shows both geometric and Poisson prior length distributions for a mean length of 3.0. Since the minimum length of a production in our case is one (not zero), the encoding of a production of length k takes $\log p(k-1;\mu)$ bits, plus $\log |\mathcal{N}|$ bits for each of the k nonterminals.

In either case, the total description length $\mathrm{DL}(M_S)$ for the grammar productions is computed, and a prior is assigned to M_S such that

$$\log P(M_S) = -\mathrm{DL}(M_S) \quad .$$

The criterion used for model evaluation is the posterior of the model *structure*, as discussed in Section 2.5.7. It is approximated using the same Viterbi method as described for HMMs (Section 3.4.3). This again has the advantage that the posterior is decomposed into a product of terms, one for each grammar nonterminal. Changes to the posterior are computed efficiently by recomputing just the terms pertaining to nonterminals affected by the merging or chunking operation.

4.3.5 Search strategies

The question of how to search efficiently for good sequences of merging operators becomes more pressing in the case of SCFGs. The main reason is that the introduction of the new operator, chunking, creates a more complex topology in the search space. In addition, the evaluation of chunking step is not directly comparable to merging, as chunking does not have a generalizing effect on the grammar (it can only affect the prior contribution to the posterior).

We have experimented with several search strategies for SCFG learning, discussed below. Clearly more sophisticated ones are possible, and await further study.

Best-first search This is the straightforward extension of our approach to merging with HMMs. All operator types and application instances are pooled for the purpose of comparison, and at each step the locally best one is chosen. This is combined with the simple linear look-ahead extension described in Section 3.4.5 to help overcome local maxima.

This simple approach often fails because chunking typically has to be followed by several merging steps to produce an overall improvement. The look-ahead feature often doesn't help here as other chunks get in the way between a chunking step and the 'right' successive merging choices.

Multi-level best-first search One possible solution to the above problem is to make the search procedure aware of the different nature of the two operators, by constraining the way in which they interact. Empirically, the following simple extension of the best-first paradigm seems to work generally well for many SCFGs.

The basic idea is that the search operates on two distinct levels, associated with merging and chunking, respectively. Search at the merging level consists of a best-first sequence of merging steps (with look-ahead). Search at the second level chooses the locally best chunking step, and then proceeds with a search at level 1. (Clearly, this approach could be generalized to any number of search levels).

Notice that in this approach, the chunking steps are *not* evaluated by trying an exhaustive sequence of merges following each possible choice. This would entail an overhead that is quite significant even in small cases.

Beam search In a beam search the locality of the search is relaxed by considering a pool a relatively good models simultaneously, rather than only a single one as in best-first search. In Section 3.3 we remarked that beam-search for HMMs seems to only very rarely give worthwhile improvements over the best-first approach.

However, beam search can produce significantly improved search results for many SCFGs, precisely because of the interaction between the different search operators and the impact this has on the proper evaluation of choices. If the beam width is made sufficiently large, the effects of combinations of merging and chunking will be assessed correctly, even if the two types of operators are not treated specially. (Adding a multi-level approach here might result in further improvements to efficiency and/or results, but hasn't been investigated yet.)

For concreteness we give a brief account of the beam search algorithm used, especially since the open-ended nature of the search (absence of a goal state) makes it different from standard beam search algorithms found in the literature. The *beam* is a list of *nodes* ordered by descending evaluation score. In our case, a node corresponds to a model (grammar), and the evaluation function is its posterior probability. Nodes can be either *expanded* or *unexpanded*, depending on whether successor nodes have been generated from them, using the search operators. When a node is expanded its descendants are inserted into the beam as unexpanded nodes, unless they are already found there. A single step in the beam search consists of expanding all unexpanded nodes in the current beam, using all available operators.

The scope of the beam search is determined by two parameters: The *beam depth* is the maximum total number of nodes in the beam, whereas the *beam width* gives the number of unexpanded nodes allowed in the beam. During expansion of the beam, low-scoring nodes are truncated from the beam so as to not exceed either width or depth. (Alternatively, one may also limit nodes in the beam to those scoring within a certain tolerance of the current best node.) The combination of conditions delimiting the elements of the beam are also known as the *beam criterion*.

The search terminates when no unexpanded nodes satisfy the beam criterion, i.e., only expanded nodes remain in the beam. The first, best-scoring node from the beam is returned as the result of the search.

Beam search as described here is a generalization of the (one-level) best-first search introduced in Section 3.4.5: a beam search of depth d and width one is equivalent to a best-first search with d steps of lookahead.

Search in grammar spaces raises the question of how the equivalence of two models should be determined efficiently. This is necessary to avoid duplicate models from crowding out worthwhile contenders in the beam. Duplicates are generated pervasively, as the same operators, such as merging, applied in different order often yield identical results. To address this problem for SCFGs and similar types of models we use a two-pronged approach. First, efficient methods for computing the posteriors of models, without necessarily computing the full models themselves are applied, using incremental evaluation strategies as described in Section 3.4.3. If two model have different posterior probabilities they must be structurally different. Secondly, if necessary, we compute a hash function of the CFG structure, which is a pseudo-random number that depends only on the structure of the productions, but not on their order or the names of the nonterminals used. If two grammars yield the same hash code they are considered identical for the purposes of beam search. This leaves a small probability that a model might mistakenly be discarded.[6]

[6]If the hash function were optimal, that probability would be 2^{-28} in the current implementation.

4.3.6 Miscellaneous

4.3.6.1 Restricted chunking

We already observed that unrestricted chunking can produce arbitrary CFG structures. For practical purposes, however, the set of potential chunks needs to be restricted to avoid generating an infeasibly large number of hypothesis in each search step. Specifically, the following restrictions can optionally be imposed.

1. No null productions. These would result from proposing empty chunks.

2. No unit (or chain) productions. These would be the result of singleton chunks.

3. A sequence of nonterminals (of any length exceeding 1) needs to occur at least twice in the grammar to be a candidate for chunking.

4. A chunk is replaced wherever it occurs, as opposed to choosing only a subset of the occurrences.

It is not known what sort of global constraint the last two restrictions place on the grammars that can be inferred, since both make reference to the form of an intermediate grammar hypothesis, which depends on the actually occurring samples and the dynamics of the search process.

4.3.6.2 Chunking undone

Occasionally a chunking operation and the nonterminals created for it become superfluous in retrospect, because only one occurrence of the nonterminal remains in the grammar as a result of productions merging. In this case it is beneficial to *undo* the chunking operation, a step we call *unchunking*. Note that the final outcome in such cases could have been achieved by not choosing to chunk in the first place, but unchunking provides a trivial and convenient way to recover from chunks that seem temporarily advantageous.[7]

4.3.6.3 Efficient sample incorporation

The simple extension of the batch merging procedure to an incremental, on-line version was already discussed for the HMM case (Section 3.3.5), and can be applied unchanged for SCFGs. This includes the use of a prior weighting factor λ to control generalization and prevent overgeneralization during the early rounds of incremental merging (Section 3.4.4). Incremental merging is the default method used in all the experiments reported below, unless otherwise noted.

As a result of incremental nonterminal merging, a sequence of nonterminals that was previously the subject of chunking can reappear. In that case the chunking operation is re-applied and the previously allocated LHS nonterminal is used in replacing the re-occurring sequence. This special form of the chunking operator is known as *rechunking*.

Various additional strategies are possible in order to reduce the number of new nonterminals created during sample incorporation, thereby reducing subsequent merging work. The drawback of all these

[7] The unchunking operation was adapted from Cook *et al.* (1976) after we had noticed the similarity between the two approaches (see the discussion in Section 4.4.3)

approaches is that they reuse previously hypothesized grammar structures, possibly preventing the algorithm from considering better alternatives.

1. Avoid duplicate samples: incorporate duplicate samples only once, with appropriately adjusted counts. This is a trivial optimization that can never do harm.

2. Try parsing samples first before resorting to the ordinary creation of new productions. If a new sample is parsed successfully counts on the old productions are updated to reflect the new sample.[8] This method subsumes strategy 1 above. (See Section 6.5.3 for ways to efficiently handle the parsing of bracketed samples, which is needed if this method is to applied to structured samples.)

3. To save initial merging of preterminals, reuse existing preterminals where possible. This precludes the creation of grammars with ambiguity at the level of lexical productions.

4. Try to parse the new sample into string fragments using existing rules, and add only a top-level production to link these fragments to the start symbol. This subsumes both strategy 2 and strategy 3. (Section 6.5.4 describes one approach to parsing ungrammatical samples into fragments that can be used here.)

Unless noted otherwise, only strategy 2 was used in obtaining the results reported here.

4.4 Related Work

4.4.1 Bayesian grammar learning by enumeration

We already mentioned Horning (1969) as an early proponent of the Bayesian version of grammar inference by enumeration, as the principle is general enough to be applied (in theory) to any type of probabilistic grammar. Horning's focus was actually on probabilistic CFGs, and the formal device used to enumerate grammars, as well as to assign prior probabilities, was a grammar-generating grammar, or *grammar grammar*. As expected, enumeration is not practical beyond the simplest target grammars, but Horning's work is theoretically important and was one of the first to point out the use of posterior probabilities as a formalization of the simplicity vs. data fit trade-off.

4.4.2 Merging and chunking based approaches

The idea of combining merging and chunking with a hill-climbing style search procedure to induce CFG structures seems to have been developed independently by several researchers. Below is a list of those we are aware of.

[8] If the sample is ambiguous the counts could be updated for all derivation according to their respective probabilities, or using only for the Viterbi derivation. In either case the likelihood of the sample will be underestimated by the Viterbi-based computation of the posterior probability. Updating according to the Viterbi derivation should favor the creation of unambiguous grammar structures, but no detailed comparisons have been done on this issue.

Cook *et al.* (1976) exhibit a procedure similar to ours, using a somewhat different set of operators.[9] Their approach is also aimed at probabilistic SCFGs, but uses a conceptually quite different evaluation function, which will be discussed in more detail below, as it illustrates a fundamental feature of the Bayesian philosophy adopted here.

Langley (1994) discusses a non-probabilistic CFG induction approach using the same merging and chunking operators as described here, which in turn is based on that of Wolff (1978). Langley's CFG learner also alternates between merging and chunking. No incremental learning strategy is described, although adding one along the lines presented here seems straightforward. The evaluation function is non-probabilistic, but incorporates several heuristics to control data fit and a bias towards grammar 'simplicity,' measured by the total length of production RHSs. A comparison with our Bayesian criterion highlights the considerable conceptual and practical simplification gained from using probabilities as the universal 'currency' of the evaluation metric.

The present approach was derived as a minimal extension of the HMM merging approach to SCFGs (see Section 3.5 for origins of the state merging concept). As such, it is also related to various induction methods for non-probabilistic CFGs that rely on structured (parse-tree skeleton) samples to form tree equivalence classes that correspond to the nonterminals in a CFG (Fass 1983; Sakakibara 1990). As we have seen, merging alone is sufficient as an induction operator if fully bracketed samples are provided.

4.4.3 Cook's Grammatical Inference by Hill Climbing

Cook *et al.* (1976) present a hill-climbing search procedure for SCFGs that shares many of the features and ideas of ours. Among these is the best-first approach, and an evaluation metric that aims to balance 'complexity' of the grammar against 'discrepancy' relative to the target distribution. A crucial difference is that only the relative frequencies of the samples, serving as an approximation to the true target distribution, are used.

Discrepancy of grammar and samples is evaluated by a metric that combines elements of the standard relative entropy with an ad-hoc measure of string complexity. Complexity of the grammar is likewise measured by a mix of rule entropy and rule complexity.[10] Discrepancy and complexity are then combined in a weighted sum, where the weighting factor is set empirically (although the induction procedure is apparently quite robust with respect to the exact value of this parameter).

To see the conceptual difference to the Bayesian approach, consider the introductory example from Section 4.3.2. The four samples $(ab, aabb, aaabbb, aaaabbbb)$ observed with relative frequencies (10, 5, 2, 1) are good evidence for a generalization to the target grammar that generates $\{a^n b^n, n > 0\}$. However, if the same samples were observed with hundred-fold frequencies (1000, 500, 200, 100), then the hypothesis $\{a^n b^n, n > 0\}$ should become rather unlikely (in the absence of any additional samples, such as a^5b^5, a^6b^6, etc.) Indeed our Bayesian learner will refrain from this generalization, due to the 100-fold increased log

[9] Thanks to Eugene Charniak for pointing out this reference, which seems to be less well-known and accessible than it deserves.

[10] The exact rationale for these measures is not entirely clear, as the complexities of strings are determined independently of the underlying model, which is inconsistent with the standard information theoretic (and MDL) approach.

likelihood loss in that case. Algorithms based on just the relative frequencies of samples, on the other hand, will be indifferent to this change in absolute frequencies. Also, the Bayesian approach gives an intuitive interpretation to the weighting factor that is useful in practice to globally balance the complexity and discrepancy terms (Section 3.4.4).

Cook *et al.* (1976) propose a larger set of operators which partly overlaps with our merging and chunking operations. Chunking is known under the name 'substitution.' Merging is not directly available, but similar effects can be obtained by an operation called 'disjunction,' which creates new nonterminals that expand to one of a number of existing nonterminals. They also have special operations for removing productions which are subsumed or made redundant by others. These can mostly me emulated with merging, although explicit testing for, and elimination of redundant productions is also useful in our algorithm, since it shortcuts combinations of induction steps (i.e., they are *macro operators* in search parlance).[11]

Cook *et al.* (1976) evaluate their algorithm using a number of benchmark grammars; these will be reexamined below using our Bayesian algorithm.

4.5 Evaluation

The merging algorithm for SCFGs has been evaluated in a number of experiments. These fall naturally into two broad categories: simple formal languages and various grammar fragments modeling aspects of natural language syntax.

4.5.1 Formal language benchmarks

This group of test grammars has been extracted from the article by Cook *et al.* (1976) discussed in Section 4.4. Except for the last two, they represent examples of simple context-free formal languages as they are typically given in textbooks on the subject.

The main advantage of using this same set of grammars is that the results can be compared. Since the two algorithms (ours and Cook's) have similar underlying intuitions about structural grammar induction we expect similar results, but it is important to verify that expectation.

Experimental setup To replicate the examples given by Cook, we used the following procedure. The samples given in the paper are used unchanged, except that sample probabilities were converted into counts such that the total number was 50 for each experiment. These were then incorporated into initial grammars and merged in batch mode. Our SNF grammar format could introduce a subtle inductive bias not present in Cook original experiments. The merging procedure was therefore constrained to always maintain a one-to-one correspondence between terminals and preterminals, effectively making terminals redundant and letting preterminals function as the true terminals, in terms of which arbitrary productions are now possible. Chunking of single nonterminals was allowed.

[11] Notice how repeated merging and chunking effectively eliminate redundant productions in the example of Section 4.3.2.

Language	Sample no.	Grammar	Search
Parentheses	8	$S \rightarrow () \mid (S) \mid SS$	BF
a^{2n}	5	$S \rightarrow aa \mid SS$	BF
$(ab)^n$	5	$S \rightarrow ab \mid SS$	BF
$a^n b^n$	5	$S \rightarrow ab \mid aSb$	BF
wcw^R, $w \in \{a, b\}^*$	7	$S \rightarrow c \mid aSa \mid bSb$	BS(3)
Addition strings	23	$S \rightarrow a \mid b \mid (S) \mid S + S$	BS(4)
Shape grammar	11	$S \rightarrow dY \mid bYS$ $Y \rightarrow a \mid cY$	BS(4)
Basic English	25	$S \rightarrow$ I am A | he T | she T | it T $\rightarrow$ they V | you V | we V $\rightarrow$ this C | that C $T \rightarrow$ is A $V \rightarrow$ are A $Z \rightarrow$ man | woman $A \rightarrow$ there | here $C \rightarrow$ is a Z | ZT	BS(3)

Table 4.1: Test grammars from Cook *et al.* (1976).

Sample number refers to sample types, not tokens. Cook's samples typically represent the highest probability strings from each sample grammar, with relative frequencies matching the probabilities. Sample counts were scaled to total 50 for all experiments. *Grammar* is the grammar found (identical to the one given by Cook). To save space the grammars are written in Cook's notation, not the SNF used in our experiments. *Search* is the simplest search method needed to obtain the result, either best-first (BF), or beam search with width n (BS(n)).

'Shape grammar' refers to a CFG used by Lee & Fu (1972) to model two-dimensional chromosome pictures (with samples from that paper). 'Basic English' are the first 25 sentences from a language teaching textbook (Richards & Gibson 1945).

The priors used were the 'standard' ones: a symmetrical Dirichlet distribution over each set of rule probabilities with $\alpha_0 = 1.0$, and the simple description length prior on the grammar structure. No extra global weighting on the prior was applied ($\lambda = 1.0$).

Results The grammars found by the merging algorithm matched the ones given by Cook in all cases. Table 4.1 gives a summary of the languages, samples and the grammars found.

The exact match of the results is somewhat remarkable since the evaluation function used in our algorithm differs significantly from that used by Cook (see the discussion in Section 4.4.3). The search operators and procedure are also different, although based on similar intuitions. This seems to indicate that the languages in question are all fairly robust with respect to any reasonable formalization of the trade-off between data fit and grammar complexity.

Interestingly, the number of samples chosen, 50, was important in matching Cook's result. Doubling the number caused the resulting grammars to be less general in several cases, and giving significantly less samples (10) would produce over-general results. We already discussed why an evaluation function should, in fact, depend on the actual number of samples, not just their relative frequencies, and one can conclude that Cook's evaluation function is tuned to be roughly equivalent to the Bayesian posterior for a sample count around 50.

Palindromes The simple palindrome language $\{ww^R, w \in \{a, b\}^+\}$ has proven to be a relatively difficult benchmark language in several investigations into CFG learning algorithms. Cook *et al.* (1976) discuss it briefly as a language beyond the reach of their search operators, noting that it would require a chunking operator that does not blindly replace all occurrences of a chunk. To learn this grammar, one has to start chunking aa and bb, sequences that typically also occur in non-center position, thereby misleading a strictly greedy learning strategy.

Pereira & Schabes (1992) use the same language to show that standard SCFG estimation does very poorly in finding a grammar of the right form when started with random initial parameters. The results of estimation can be improved considerably by using bracketed samples instead (although the resulting grammar still gives incorrect parse structure to some strings).

Indeed our algorithm fails to find a good grammar for this language using only best-first search, due to the limitations of the chunking operation cited by Cook.[12] However, the result improved when applying the more powerful beam search (beam width 3).

```
S --> A A
  --> B B
  --> A S A
  --> B S B
  --> S S    ***
A --> a
B --> b
```

This is almost a perfect grammar for the language, except for the production marked by `***`. This production is redundant (not strictly required for the derivation of any of the samples) and can be eliminated by a simple Viterbi reestimation step following the search process. (The Viterbi step is meant to correct counts that have become inaccurate due to the optimistic update procedure during merging.)

While this result in itself is not very significant it does show that the simple merging operators become considerably more powerful when combined with more sophisticated search techniques, as expected.

Incidentally, the palindrome language becomes very easy to learn from bracketed samples, using merging only.

[12] For concreteness, we used the same training setup as for the easier palindrome language with center symbol (fifth in Table 4.1). The samples and their number where identical except for removal of the center marker c.

4.5.2 Natural language syntax

In this section we show examples of applying the Bayesian SCFG merging algorithm to simple examples of natural language syntax. A strong caveat is in order at this point: these experiments do not involve actual natural corpus data. Instead, they make use of artificial corpora generated from grammars that are supposed to model various aspects of natural languages in idealized form, and subject to the inherent constraints of context-free grammars.

The main goal in these experiments will be to test the ability of the algorithm to recover specific typical grammatical structures from purely distributional evidence (the stochastically generated corpora). Applying the algorithm to actual data involves additional problems that will be discussed towards the end.

As indicated in Section 4.3.4, we used a prior for production lengths corresponding to a Poisson distribution. The prior mean for the length was set to 3.0.

Lexical categories and constituency The following grammar will serve as a baseline for the following experiments. It generates simple sentences based on transitive and intransitive verbs, as well as predications involving prepositional phrases. Unless otherwise indicated, all productions for a given LHS have equal probability.

```
S --> NP VP
NP --> Det N
VP --> Vt NP
   --> Vc PP
   --> Vi
PP --> P NP
Det --> a
    --> the
Vt --> touches
   --> covers
Vc --> is
Vi --> rolls
   --> bounces
N  --> circle
   --> square
   --> triangle
P  --> above
   --> below
```

The corresponding corpus contains 100 randomly generated sentences, including repetitions. (The total number of distinct sentences generated by the grammar is 156.) The following samples illustrate the range of allowed constructions:

```
the circle covers a square
a square is above the triangle
a circle bounces
```

The corpus was given as input to incremental, best-first merging. The algorithm produced the eventual result grammar after having processed the first 15 samples from the corpus.

The result grammar is weakly equivalent to (generates the same sentence as) the target, and has identical lexical rules. However, the sentence-level productions are less structured:[13]

```
S --> NP VC P NP        (28)
  --> NP VI             (39)
  --> NP VT NP          (33)
NP --> DET N            (161)
```

The algorithm was thus successful in grouping terminals into appropriate lexical categories and identifying the pervasive NP constituents. The log posterior probability of this grammar (-230.30769) is slightly below that of the target grammar (-230.1136).

However, a more extensive beam-search (width 30) finds an alternative grammar exhibiting a deeper phrase structure:

```
S --> NP VP             (100)
VP --> VI               (39)
   --> X NP             (61)
X --> VT                (33)
  --> VC P              (28)
NP --> DET N            (161)
```

This structure turns out to have a somewhat higher log posterior probability than the target (-228.658). Chunking (`VC P`) is prefered to the standard (`P NP`) because it allows merging the two VP expansions involving `VC` and `VT`, respectively.

Interestingly, the importance of the prior distribution for the production lengths is already evident in this experiment. Were it not for the Poisson prior length distribution, the flat productions in the first grammar above would actually yield a higher posterior probability than the target grammar.

Recursive embedding of constituents To get PP constituents based only on distributional evidence, the grammar can be enriched, e.g., with topicalized PPs and PPs embedded in NPs. The changed productions are

```
S --> NP VP             [3/5]
  --> PP COMMA NP Vi    [1/5]
  --> PP Vc NP          [1/5]
NP --> Det N            [3/4]
   --> Det N PP         [1/4]
COMMA --> ,
```

This extends the range of sentence to samples such as

```
above a square is the square
the circle below the triangle below a circle touches the triangle
above a triangle , a circle rolls
below a square , the triangle above the square below the square bounces
```

[13] Induced grammars in this section are notated with nonterminal labels that chosen from the target grammar where possible, so as to enhance readability. The actual nonterminal names are of course irrelevant to the algorithm.

After processing 100 random samples of this grammar (which now generates an unbounded number of sentences), the following phrase structure rules are found.

```
S --> NP VC PP           (22)
  --> NP VI              (10)
  --> NP VT NP           (29)
  --> PP COMMA NP VI     (19)
  --> PP VC NP           (20)
PP --> P NP              (112)
NP --> DET N             (241)
   --> NP PP             (51)
```

The induced rule structure differs in two aspects from the target. Firstly, no VP phrase structure in inferred. This is not surprising since each of the VP realizations appears only in a single rule, i.e., there is no generalization or greater succinctness to be gained from positing a VP constituent. Secondly, the embedding of PP in NP is realized in two, rather than one recursion: one left-recursive through the NP rule, and one tail-recursive through the PP rule. This redundancy also leads to a slightly suboptimal score compared to the target grammar. Still, the grammar found is clearly weakly equivalent to the target.

Agreement Another pervasive phenomenon in natural languages is agreement, whereby two lexical items or constituents are forced to share one of more (abstract) features such as number, gender, case marking, etc. For our purposes we can construe the phonologically motivated alternation of English determiners (‘a’ and ‘the’ versus ‘an’ and ‘thee’) as a case of NP-internal agreement. A minimal extension of the baseline grammar to this end is the following (only nonterminals that differ from the baseline are shown):

```
NP --> Det1 N1
   --> Det2 N2
Det1 --> a
     --> the
Det2 --> an
     --> thee
N1 --> circle
   --> square
   --> triangle
N2 --> arc
   --> octagon
```

After merging 40 random samples, these productions were in fact generalized correctly, while the remainder of the grammar was identical to the one learned before from the baseline corpus.

It should be noted that context-free grammars are not particularly suited to the description of agreement phenomena, especially if the agreeing constituents are separated by material not affected by the agreement. A CFG description in this case has to replicate syntactic categories up to the smallest enclosing constituent level.[14]

[14] The standard solution to this problem is to factor agreement into a separate description, usually using a feature-based formalism.

The problem is demonstrated by the following minimal extension of the baseline grammar to incorporate singular and plural NPs and enforce number agreement between main verb and subject NP.

```
S --> NP_SG VP_SG
  --> NP_PL VP_PL
NP --> NP_SG
   --> NP_PL
NP_SG --> DET N_SG
NP_PL --> DET N_PL
VP_SG --> VI_SG
      --> VT_SG NP
      --> VC_SG PP
VP_PL --> VI_PL
      --> VT_PL NP
      --> VC_PL PP
PP --> P NP
DET --> the
N_PL --> circles | squares | triangles
N_SG --> circle | square | triangle
P --> above | below
VC_SG --> is
VC_PL --> are
VI_SG --> bounces | rolls
VI_PL --> bounce | roll
VT_SG --> covers | touches
VT_PL --> cover | touch
```

The grammar now generates sentences including

```
the circle bounces
the circles bounce
the square is below the triangles
the squares are below the triangles
```

This extensions of the scope of the language comes at a considerable price: the categories N, NP, VP, Vt, Vc, Vi all have to be duplicated to encode the number feature.

Subjecting a 200-sentence sample to the merging algorithm exposes an additional, unexpected problem. The grammar learned is weakly equivalent to the target grammar; in particular, it contains all of the lexical categories in the target grammar. However, the phrase structure rules found are quite different: they group the verbs and prepositions with the subject, rather than the object NPs.

```
S --> NP_SG VI_SG       (27)
  --> NP_PL VI_PL       (37)
  --> X NP_SG           (76)
  --> X NP_PL           (60)
X --> NP_SG VT_SG       (37)
  --> NP_PL VT_PL       (33)
  --> NP_SG VC_SG P     (37)
  --> NP_PL VC_PL P     (29)
NP_SG --> DET N_SG      (177)
NP_PL --> DET N_PL      (159)
```

This grammar has a marginally lower posterior probability than the target grammar. The reason why even fairly wide beam search consistently finds this phrase structure, rather than the traditional one, is subtle, and has to do with the inherent representational problems for agreement phenomena in CFGs.

Because agreement has to be mediated by duplication (or rather, non-merging) of otherwise identical nonterminal symbols up to the smallest phrase level enclosing the agreeing elements, it is advantageous to group agreeing, rather than non-agreeing parts of the syntax. This strategy minimizes the number of extra nonterminals required. The two fundamental chunking alternatives found in the example (verbs with subject, vs. with object) only regain equal description length once all other generalizations have been found. Thus a localized search will always prefer the subject-verb grouping.

Presumably a actual natural language learner would have access to other cues that favor chunking verbs with their objects. This can at least be simulated by modifying the samples to contain partial bracketings, e.g.,

```
the square (is above the triangle)
the triangles (are below circles)
```

It not necessary to add partial bracketing to all samples, since some partially bracketed samples are enough to make chunking of the remaining ones in the appropriate way the best-scoring decision. The following grammar was learned from the same 200-sentence corpus as before, modified so that that 50% of the samples had explicit VP bracketing. As expected, this leads to the traditional phrase structure, including the necessary nonterminal duplication to account for agreement.

```
S --> NP_SG VP_SG          (101)
  --> NP_PL VP_PL          (99)
VP_SG --> VI_SG            (27)
      --> VT_SG NP_SG      (19)
      --> VT_SG NP_PL      (18)
      --> VC_SG PP         (37)
VP_PL --> VI_PL            (37)
      --> VT_PL NP_SG      (17)
      --> VT_PL NP_PL      (16)
      --> VC_PL PP         (29)
PP --> P NP_SG             (40)
   --> P NP_PL             (26)
NP_SG --> DET N_SG         (177)
NP_PL --> DET N_PL         (159)
```

Incidentally, the reason the productions

```
NP --> NP_SG
   --> NP_PL
```

are not found is that chunking always replaces all occurrences of a chunk. As a result, the operations chunk(NP_SG) and chunk(NP_PL) would interfere with the productions that handle the agreement, and are rightfully rejected.

A.

```
S --> NP VP
VP --> VerbI
   --> VerbT NP
NP --> Art Noun
   --> Art AP Noun
AP --> Adj
   --> Adj AP
Art --> the
VerbI --> ate | slept
VerbT --> saw | heard
Noun --> cat | dog
Adj --> big old
```

B.

```
S --> NP VP
VP --> Verb NP
NP --> Art Noun
   --> Art Noun RC
RC --> Rel VP
Verb --> saw | heard
Noun --> cat | dog | mouse
Art --> a | the
Rel --> that
```

Figure 4.2: Example grammars from Langley (1994).

4.5.3 Sample ordering

We conclude with the two pseudo-natural language examples from Langley (1994), where a hill climbing learner for non-probabilistic CFGs is studied (see Section 4.4). The point here will be that the *ordering* of samples during incremental processing can have a considerable effect on the outcome, or on the speed of the learning process.

The grammars shown in Figure 4.2 exhibit two types of recursion. Grammar A contains an NP rule for an arbitrary number of adjectives, such as in

```
the old big cat heard the big old big dog
```

Grammar B is similar but allows embedded relative clauses instead of adjectives. It yields sentences such as

```
a mouse heard a dog that heard the mouse that heard the cat
```

For the purpose of our experiment uniform probabilities were added to the original non-probabilistic CFG productions, and were used to generate 100 random samples each (sample A and B, respectively). The samples were presented in one of two conditions: random order, or in order of increasing sentence length (number of words).[15] Both samples were processed by incremental merging and chunking.

Sample A ordered by sentence length resulted in a grammar that contained three redundant rules that could be removed by reestimation. The final grammar was:

```
S --> NP VI            (57)
  --> NP VT NP         (43)
NP --> DET N           (67)
   --> DET NP1         (76)
NP1 --> ADJ N          (76)
    --> ADJ NP1        (71)
DET --> the            (143)
N --> cat              (73)
```

[15] The order among samples of the same length remained random.

```
  --> dog           (70)
ADJ --> big         (75)
    --> old         (72)
VI --> ate          (29)
   --> slept        (28)
VT --> heard        (24)
   --> saw          (19)
```

This is essentially the phrase structure of the target grammar, except that the recursion in noun phrases is implemented differently.[16]

On the other hand, when given the unordered samples, the algorithm produced a grammar that contained nine redundant rules. However, the same grammar as above was found by removing these rules, restarting the search and pruning three more redundant rules.

With ordered samples, 16 of the 100 samples required creation of new rules, and 212 merging steps were necessary to produce the final grammar, taking 49 seconds total CPU time. The unordered samples, on the other hand, lead to new productions for 20 samples, 1374 merges, and took 174 seconds.

For sample B, the outcome was similar. Ordering the samples by length resulted in the following grammar, an acceptable rendering of the original.

```
S --> NP VP          (100)
VP --> V NP          (295)
NP --> DET N         (395)
   --> NP RC         (195)
RC --> REL VP        (195)
DET --> a            (196)
    --> the          (199)
N --> cat            (137)
  --> dog            (118)
  --> mouse          (140)
REL --> that         (195)
V --> heard          (152)
  --> saw            (143)
```

A comparison experiment on the unordered list of samples had to be aborted due to excessively long run time.

There are several reasons why ordering samples by length for incremental merging leads to improved speed and accuracy.

- 'Similar' samples, which eventually lead to a single generalization are presented closer together, thereby reducing the time it takes to hypothesize and accept a generalization.

- As a result, the number of samples and corresponding states that are not yet fully merged into the model structure are minimized, reducing the search space overall.

[16] A beam search on the same data produced a grammar that was actually better than the target grammar, i.e., it was weakly equivalent, produced essentially the same phrase structure and used shorter rules.

- Rules learned from smaller samples tend to be useful in structuring larger samples (but not the other way around). Thus the analyses of the previous samples can effectively guide the search for merges based on the longer, more recent ones.

An interesting related result from non-probabilistic language induction is that strictly (lexicographically) ordered sample presentation makes the learning problem for certain classes of grammar provably easier (Porat & Feldman 1991).

4.5.4 Summary and Discussion

The above artificial examples show that SCFG learning based purely on distributional evidence and the generic Bayesian simplicity bias can correctly identify several of the common linguistic structures. It also shows that this limited evidence can be misleading, especially when it comes to finding the 'right' phrase structures. This is hardly surprising, given that a lot of the cues for human judgments of linguistic structure presumable come from other sources, such as the semantic referents of the syntactic elements, phonological cues, morphological markers, etc. (Morgan *et al.* 1987).

In a brief experiment, we applied our algorithm to a 1200 sentence corpus collected with the BeRP speech system (Jurafsky *et al.* 1994a). The algorithm produced mostly plausible lexical categories and a large number of chunks corresponding to frequent phrases and collocations. However, the generalization achieved was nowhere near what would be required for a sufficient coverage of new data. The problems can partly be addressed by simple preprocessing steps, such as tagging of lexical items using standard probabilistic approaches that achieve reasonable performance on data of this sort (Kupiec 1992b). Non-traditional phrase structuring may not be a problem if sentence-level generalization is the main goal for an application. Also, bracketing may be induced separately using bracketing models trained from structured data (Brill 1993). We plan to investigate such hybrid strategies in the future.

Chapter 5

Probabilistic Attribute Grammars

5.1 Introduction

This chapter describes an extension of stochastic context-free grammars (Chapter 4) that allows the probabilistic modeling of simple kinds of *features* or *attributes* attached to the nonterminals in a grammar. For example, we can use these attributes to represent limited semantics of sentences and their constituents. We will call this extension *probabilistic attribute grammars (PAGs)*, although the particular formulation chosen here is only one such possible extension, and is minimal in several ways.

The learning problem associated with PAGs is as follows. Samples from a PAG language are pairs of sentence strings and attribute-value sets. (The attributes and their values implicitly describe the root nonterminal of the sentence). As with other types of grammars, the goal is to find a PAG that generates the presented samples, possibly with generalizations extracted from the samples. Since PAGs are probabilistic models their quality can be evaluated using the same quantities (likelihoods and prior probabilities) as usual.

We start by defining the PAG extension to SCFGs (Section 5.2). A parallel extension of the SCFG merging algorithm so that it can infer simple PAGs is given in Section 5.3. Experiments drawn from the L_0 language learning scenario introduced in Chapter 1 are described in Section 5.4.

Due to its simplicity, the current PAG formalism can be little more than an illustration of how the standard probabilistic models and the learning-by-merging principle can be generalized to new types of grammars. Section 5.5 discusses limitations and problems with the current approach and points out possible remedies. Section 5.6 summarizes the main points.

5.2 Probabilistic Attribute Grammars

Attribute grammars are a familiar extension of non-probabilistic CFGs known from compiler theory and elsewhere (Aho *et al.* 1986). Attributes are 'registers' or 'slots' attached to the nonterminal nodes in a context-free parse tree, and can hold 'values' for various purposes, typically pertaining to the semantics of the language. The context-free productions are augmented with attribute specifications that determine how a nonterminals attributes are computed as a function other nonterminals' attribute values and terminals.[1]

Many linguistic formalisms based on context-free grammars employ a related concept, typically known by the name of 'features.' Like attributes, these are attachments to the nonterminals in the grammar, but their values and their specifications are usually more restricted, in the form of directed acyclic graphs and unification equations (Shieber 1986).

The attribute grammar model used here is a much simplified variant of either of these two traditional CFG extensions, with a probabilistic component added to yield the usual notions of sample probability, likelihood, etc. We will use the terms *attribute* and *feature* interchangeably.

5.2.1 Definitions

The model can be characterized as follows.

Definition 5.1 A *probabilistic attribute grammar (PAG)* is a stochastic context-free grammar with the following additional provisions.

a) Each nonterminal $X \in \mathcal{N}$ in a SCFG derivation has an associated finite set a features from a set $\mathcal{F}$. A feature $f \in \mathcal{F}$ is a function that assigns an element from the finite set of *feature values* $\mathcal{V}$ to the nonterminals it is defined on. Following the traditional precedents mentioned above, we denote the fact that feature f on nonterminal X has value v by

$$X.f = v$$

b) An extended context-free production specifies the feature values attached to the LHS nonterminal in terms of either constant values or the features of RHS nonterminals. Thus, a production

$$X \rightarrow Y^{(1)}Y^{(2)}\ldots Y^{(k)}$$

can have feature specifications

$$X.f = v$$

for some $f \in \mathcal{F}$ and $v \in \mathcal{V}$, or

$$X.f = Y^{(i)}.g$$

for some i, $1 \leq i \leq k$, and $g \in \mathcal{F}$. The latter specification determines that the value of $X.f$ is whatever the value of feature g on $Y^{(i)}$ is.

c) Feature equations have associated probabilities, written

$$P(X.f = v)$$

[1]Certain ordering constraints are typically imposed to guarantee that these attribute values can actually be computed effectively and efficiently.

or

$$P(X.f = Y^{(i)}.g)$$

All feature specifications for a certain LHS feature $X.f$ are mutually exclusive, i.e., to assign a LHS feature one chooses probabilistically among all feature values and RHS features. Therefore all probabilities involving $X.f$ on the LHS must sum to unity.

Notation It is important in this context to be able to distinguish between a nonterminal *type* and a nonterminal *instance* in a derivation (i.e., as node in a parse tree). Attributes and their specifications always refer to nonterminal instances. However, to simplify the notation we can rely on the nonterminal names as they appear in the context-free production where possible without ambiguity. Where this is not sufficient, superscripts in parentheses distinguish multiple occurrences of the same nonterminal type. Thus,

$$\begin{aligned} X \quad &\rightarrow \quad YZY \\ & \quad\quad X.f = Y^{(1)}.x \\ & \quad\quad X.g = Y^{(2)}.y \end{aligned}$$

equates $X.f$ with the x value on the first Y node, and $X.y$ with the y value on the second Y node.

The above definition of PAGs already suggests a straightforward derivation model. A derivation proceeds in two phases: first, a sentence string is generated by the SCFG in the usual top-down fashion; second, a probabilistic feature assignment takes place bottom-up, by consulting the feature equations and their probabilities as determined by the SCFG derivation. The features assigned to the root nonterminal are then by definition the features of the sentence as a whole.

Definition 5.2 Let $M = (M_S, M_F)$ be a PAG, with the SCFG M_S as its set of context-free productions, and M_F as the set of feature equations. Also, $x \in \Sigma^*$ is a string and $\mathbf{f} = \{f_i = v_i, i = 1, \ldots, n\}$ is a set of feature-value assignments, where $f_i \in \mathcal{F}$, $v_i \in \mathcal{V}$ and the f_i are pairwise distinct.

a) Let $d_S = (S \Rightarrow \nu_2 \Rightarrow \ldots \nu_k \ldots x)$ be a derivation of x in M_S (Definition 4.2). Then a feature derivation d_F based on d_S is given by choosing exactly one feature equation from M_F for each LHS feature $X.f$, for all expansions $X \rightarrow \lambda$ making up d_S. The derivation d_F is said to *yield* the feature assignment **f** if the feature-value pairs in **f** are a subset of those obtained by propagating values from RHS nonterminals to LHS nonterminals in the obvious way as implied by the feature equations chosen in d_F.

b) The probability of the feature derivation d_F given the string derivations d_S, $P(d_F|d_S; M)$, is the product of the probabilities of all equations chosen.

c) The *feature probability* $P(\mathbf{f}|d_S; M_F)$ given a string derivation d_S is the sum of all $P(d_F|d_S; M_F)$ such that d_F yields **f**.

c) The *string-feature probability* $P(x, \mathbf{f}|M)$ of a string/feature assignment pair $(x, \mathbf{f})$ is

$$\sum_{d_S} P(\mathbf{f}|d_S; M_F)P(d_S|M_S)$$

where d_S ranges over all context-free derivations yielding x.

Decomposing the stochastic derivation process in this way makes for a convenient definition of the associated joint probabilities, but also rules out several of the more interesting uses of attributes. Section 5.5 discusses some alternative, computationally more complex definitions.

5.2.2 An example

A simple PAG can be easily written for the L_0 language. We consider sentences containing a single unary or binary predicate, such as

```
a circle moves
a circle is below a square
a square touches a circle
a triangle covers a square
```

The semantics of these sentences is encoded using the three attributes `tr` (trajector), `lm` (landmark), and `rel` (relation) and attribute values that correspond to the 'meanings' of the denoting words: `'circle'`, `'square'`, `'below'`, `'touch'`, etc.[2] Thus, the above samples including attributes become

```
(a circle moves, { tr = 'circle', rel = 'move' })
(a circle is below a square, { tr = 'circle', lm = 'square', rel = 'below' })
(a square touches a circle, { tr = 'square', lm = 'circle', rel = 'touch' })
(a triangle covers a square, { tr = 'triangle', lm = 'square', rel = 'cover' })
```

A PAG generating these and similar samples has a standard SCFG backbone with two types of rules: lexical productions that assign semantic constants to the denoting words, and nonterminal productions that pass these as feature values through to the root node. Since there are no semantic ambiguities in this language, all feature equations carry probability one (omitted below). The SCFG probabilities are also not shown since they could be arbitrary depending on the intended distribution.

```
S  --> NP VP
     S.tr = NP.f
     S.lm = VP.g
     S.rel = VP.h
NP --> Det N
     NP.f = N.f
VP --> Vi
     VP.h = Vi.h
   --> Vc PP
```

[2]Never mind that this notion of meaning is blatantly simplistic and English-centric: the point here is to illustrate the workings of a PAG.

```
      VP.g = PP.g
      VP.h = PP.h
    --> Vt NP
      VP.g = NP.f
      VP.h = Vt.h
Det --> a
N  --> circle
      N.f = 'circle'
    --> square
      N.f = 'square'
    ...
Vc --> is
Vi --> moves
      Vi.h = 'move'
    --> falls
      Vi.h = 'fall'
Vt --> touches
      Vt.h = 'touch'
    --> covers
      Vt.h = 'cover'
    ...
PP --> P NP
      PP.g = NP.f
      PP.h = P.h
P --> above
    P.h = 'above'
   --> below
    P.h = 'below'
   ...
```

All feature names except for the top-level ones can be chosen arbitrarily. In particular, there is no requirement that the LHS feature names coincide with RHS feature names, although this is often convenient to make the feature equations more readable.

Simple lexical ambiguities can be modeled using feature assignments with non-unit probabilities, e.g., for the nonsense word `squangle`:

```
N --> squangle
    N.f = 'square'         [0.5]
    N.f = 'triangle'       [0.5]
```

5.2.3 PAG estimation

Since PAGs are not a standard grammar formalism there is no standard estimation algorithm that sets the PAG probability parameters given a set of samples and grammar rules. However, an iterative likelihood maximization scheme based on EM techniques can be applied here as well.

Since our focus here is on merging (i.e., structure learning), rather than parameter estimation approaches, we will not give the details of this algorithm here. The basic idea is to estimate the SCFG part of the grammar separately, and then, for a given SCFG derivation to compute the co-occurrence of feature values on adjacent parse tree nodes in a way similar to the belief propagation algorithm of Pearl (1988), conditioning on the top-level feature assignments and those given by the choice of SCFG rules. Once the conditional distribution of feature values on adjacent nodes is known, the likelihood is maximized by choosing a feature equation's probability to be proportional to the number of times the two features had the same value.

While this scheme is in principle capable of finding full PAGs of the kind shown above starting from randomly initialized feature probabilities, we found that it is also very prone to local maxima and highly dependent on the initial conditions of the parameters. As expected, the problem is observed increasingly in 'deep' grammars with many intermediate of features, and hence hidden variables as part of a derivation.

Thus, the problems with the parameter estimation approach to structural learning that were previously observed with HMMs (Section 3.6.1) and SCFGs (Pereira & Schabes 1992) seem to show up even more severely in the PAG formalism. This gives further motivation to our investigation of merging-based learning approaches.

5.3 PAG Merging

The following sections describe the usual ingredients necessary to specify a merging-based learning algorithm for the PAG formalism. Not surprisingly, we can build on the corresponding components of the SCFG merging algorithm.

5.3.1 Sample incorporation

The first step in the learning algorithm is the creation of *ad hoc* rules that allow new samples to be generated. We use newly created top-level productions to generate the sample strings as before, but augment these with feature value assignments to generate the observed attribute-value pairs. Thus, a sample

```
(a circle is below a square, { tr = 'circle', lm = 'square', rel = 'below' })
```

is incorporated by creating productions

```
S --> A1 CIRCLE1 IS1 BELOW1 A2 SQUARE1 (1)
    S.tr = 'circle'                         (1)
    S.lm = 'square'                         (1)
    S.rel = 'below'                         (1)
A1 --> a                               (1)
CIRCLE1 --> circle                     (1)
IS1 --> is                             (1)
BELOW1 --> below                       (1)
A2 --> a                               (1)
SQUARE1 --> square                     (1)
```

We extend our earlier notation to associate counts with both productions and individual feature equations. Counts on feature equations indicate how often an equation has been used in feature derivations, and can be used to compute likelihoods, probability estimates, etc. (Section 3.4.3).

Notice that the features cannot be attached to the lexical productions created, since that would imply knowing the semantic contributions of each word. Hence the features are first attached to the sentence as a whole, corresponding to rote learning of sentence-meaning associations.

5.3.2 Nonterminal merging and chunking

The nonterminal merging and chunking operators introduced in Chapter 4 can be applied to PAG rules with a few modifications. Merging in the CFG part of productions operates as usual, by renaming nonterminals and possibly merging productions. Feature equations are also affected by renaming nonterminals: Whenever two or more feature equations become identical as the result of nonterminal merging, their associated counts are added as usual.

By way of example, suppose the we merge nonterminals `N1` and `N2`, starting with productions

```
NP --> Det N1       (2)
     NP.f = N1.f1        (2)
NP --> Det N2       (2)
     NP.f = N2.f2        (2)
```

The result of merge(`N1`, `N2`) = `N` is

```
NP --> Det N        (4)
     NP.f = N.f1         (2)
     NP.f = N.f2         (2)
```

Notice that the feature names `f1` and `f2` are not automatically merged; that will be accomplished by a separate operator.

Nonterminal chunking operates as before, but may require the creation of new *ad hoc* features attached to the newly created nonterminal in order to preserve any features found on the chunked nonterminals. This too is most easily seen by example:

```
S --> Det N V A N
    S.tr = N(1).f        (2)
    S.lm = N(2).f        (2)
    S.rel = V.h
```

When applying chunk(`A N`) = `NP` the `f` feature must be preserved by creating a corresponding feature on `NP`:

```
S --> NP V NP
    S.tr = NP(1).g       (2)
    S.lm = NP(2).g       (2)
    S.rel = V.h
NP --> Det N
    NP.g = N.f           (4)
```

5.3.3 Feature operators

Two new operators are necessary to perform generalizations involving feature equations. *Feature merging* is the obvious step of merging two feature names, similar to the way nonterminals are merged. Consider the grammar

```
S --> Det N V A N
    S.tr = N(1).f1
    S.lm = N(2).f2
    S.rel = V.h
N --> circle
    N.f1 = 'circle'
  --> square
    N.f2 = 'square'
```

Now features `f1`, `f2` are merged into a single feature `f`, denoted by fmerge(`f1`, `f2`) = `f`.

```
S --> Det N V A N
    S.tr = N(1).f
    S.lm = N(2).f
    S.rel = V.h
N --> circle
    N.f = 'circle'
  --> square
    N.f = 'square'
```

Feature equation counts are preserved in this step, and counts are added where equations merge as a result of the renaming of features.

The other feature operation, *feature attribution*, postulates that a certain feature value v assigned to a LHS nonterminal is actually due to a RHS nonterminal X and its feature $X.f$. The operation is written fattrib$(X, v) = f$.[3]

To illustrate, consider rules as they could have been created by sample incorporation and subsequent merging of lexical categories.

```
S --> A CIRCLE IS BELOW A SQUARE
    S.tr = 'circle'
    S.lm = 'square'
    S.rel = 'below'
S --> A SQUARE TOUCHES A TRIANGLE
    S.tr = 'circle'
    S.lm = 'square'
    S.rel = 'below'
A --> a
CIRCLE --> circle
SQUARE --> square
triangle --> triangle
IS --> is
BELOW --> below
TOUCHES --> touches
```

[3]The feature f is newly created, but an existing feature of X can effectively be reused by subsequently merging it with f.

After attributing `'square'` to the new feature `SQUARE.f` the productions become

```
S --> A CIRCLE IS BELOW A SQUARE
    S.tr = 'circle'
    S.lm = SQUARE.f
    S.rel = 'below'
S --> A SQUARE TOUCHES A TRIANGLE
    S.tr = SQUARE.f
    S.lm = 'triangle'
    S.rel = 'below'
SQUARE --> square
    SQUARE.f = 'square'
```

The choice of which feature value to attribute to which nonterminal occurrence on the RHS is nondeterministic, but heuristics can be used in practice (see below).

5.3.4 Efficient search for feature operations

Combining operators It is convenient to conceptualize the four operators described above as separate induction steps. However, in practice is more efficient to combine several of these operators due to their typical interactions. In the experiments reported below this was done so the existing two-level best-first search strategy could be used, instead of a more extensive search method.

The feature attribution and merging operators, in particular, often produce an improved posterior probability score in conjunction with certain nonterminal merging operations. Consider the rules

```
NP --> Det N1            [0.5]
     NP.f = N1.f1
   --> Det N2            [0.5]
     NP.f = 'circle'
N1 --> square
     N1.f1 = 'square'
N2 --> circle
```

Simply applying the syntactic merging operation merge(`N1`, `N2`) = `N` would result in

```
NP --> Det N             [1.0]
     NP.f = N.f                    [0.5]
     NP.f = 'circle'               [0.5]
N --> square             [0.5]
     N1.f = 'square'
  --> circle             [0.5]
```

which cuts the probability of all samples using one of these productions in half. Alternatively we might precede the merging operation by 'appropriate' feature operations, namely fattrib(`N2`, `circle`) = `f2`, followed by fmerge(`f1`, `f2`) = `f`. This gives the grammar

```
NP --> Det N             [1.0]
     NP.f = N.f                    [1.0]
```

```
N --> square            [0.5]
     N1.f = 'square'              [1.0]
  --> circle            [0.5]
     N1.f = 'square'              [1.0]
```

which preserves the likelihood of the original grammar.

The general strategy in combining feature operations with syntactic merging steps is to preserve determinism in the feature equations where possible, under the constraint that all previously generated samples can still be generated. The algorithm implementing this strategy has a striking similarity with standard feature unification. Thus, we can think of the feature operations in the example (1) as unifying a constant with the newly created feature `f2`, and (2) as unifying the features `f1` and `f2`. Unlike standard unification we can resolve feature value conflicts by splitting probabilities.

Feature attribution Although the above method can efficiently find many feature attributions, feature attributions must still be considered independently as potential search steps. A good heuristic to suggest promising candidates is correlation (or mutual information) between occurrences of nonterminals and feature values in the RHSs of productions and feature equations, respectively. Specifically, the operation fattrib(X, v) is considered if nonterminal X and feature value v have close to perfect correlation (or alternatively, large positive mutual information).

When applying an operator of type fattrib(X, v) one has to deal with the nondeterminism arising from multiple occurrences of nonterminals X in production RHSs. (Which instance of X should the value v be attributed to?) A full search of all alternative attributions would discover that some yield feature merging opportunities that preserve the model likelihood, while others do not, and chose the appropriate alternative on that basis.

When processing samples in incremental mode a simple, but effective approach is to *delay* (buffer) processing of those samples which lead to productions that give rise to ambiguity in the feature attributions. Specifically, by first processing samples with distinct feature values, attributions can be done deterministically. The ambiguities in the held-back samples are then implicitly resolved by merging with existing productions. For example,

```
S --> A CIRCLE IS BELOW A SQUARE
    S.tr = 'circle'
    S.lm = 'square'
    S.rel = 'below'
S --> A CIRCLE IS BELOW A CIRCLE
    S.tr = 'circle'
    S.lm = 'circle'
    S.rel = 'below'
```

The operations fattrib(`CIRCLE`, `circle`) is unambiguous for the first production, but not for the second one. The second production is therefore set aside, whereas the first one eventually (after several attribution steps) results in

```
S --> A CIRCLE IS BELOW A SQUARE
    S.tr = CIRCLE.f1
    S.lm = SQUARE.f2
    S.rel = 'below'
CIRCLE --> circle
    CIRCLE.f1 = 'circle'
SQUARE --> square
    SQUARE.f2 = 'square'
```

The nonterminal merging operation merge(`CIRCLE`, `SQUARE`) = `N` then triggers the merging of features `f1`, `f2`.

```
S --> A N IS BELOW A N
    S.tr = N.f
    S.lm = N.f
    S.rel = 'below'
N --> circle
    N.f = 'circle'
  --> square
    N.f = 'square'
```

The nonterminal merging step also renders the two original productions for `S` identical, and means the associated feature equations should be reconciled. To this end, the two instances of `'circle'` are implicitly attributed to the first and second `N.f` feature (as part of the combined merging operators discussed in the previous section).

5.3.5 PAG Priors

Prior probability distributions for PAGs can be constructed using the now familiar techniques. The probabilities for the features equations (involving both fixed values and RHS features) represent multinomial parameter tuples for each LHS feature, and are accordingly covered by a Dirichlet prior distribution. The feature equations themselves can be given priors based on description length. Each equation involving a constant feature value comes at a code length cost of $\log |\mathcal{V}|$ bits. Equations with features on the RHS are encoded using $\log |\mathcal{F}_X| + \log k$ bits, where X is the RHS nonterminal in question, $\mathcal{F}_X$ is the set of features attached to nonterminal X throughout the grammar (since only those need to be encoded uniquely), and k is the length of production RHS.

Using a description length prior favors feature merging and other operations that achieve more compact feature specifications by virtue of collapsing feature names and equations. Alternatively, one could make all feature specifications *a priori* equally likely[4] and rely solely on the prior of the context-free part of the grammar for a bias for smaller rules. Since feature equations cannot exist independently of context-free productions this will implicitly also bias the feature descriptions against complexity. Furthermore, redundant

[4]This is not strictly possible for a proper prior since the probability mass has to sum to one, so there has to be some sort of decay or cut-off for large grammars. However, when comparing grammars of limited, roughly equal size we can use a uniform prior as an approximation.

feature equations lead to unnecessary splitting of feature probabilities, and are thus already disfavored by the likelihood component of the posterior probability.

5.4 Experiments

5.4.1 L_0 examples

As mentioned in the introduction (Chapter 1), the motivation for the PAG extension to SCFGs came partly from the L_0 miniature language learning task proposed by Feldman *et al.* (1990). It was therefore natural to test the formalism on this problem. No description length bias on feature equations was used.

A minimal L_0 fragment for English and the associated generating grammar was already presented as the example grammar in Section 5.2.2. A version of this grammar with 3 nouns, 2 prepositions, and 2 verbs of each category was used to generate a random 100-sentence sample. All alternative productions were given equal probability. The generated sentences were then processed by incremental best-first search using the induction operators described above.

The result grammar found is weakly equivalent to the target. This includes accurate feature attributions to the various terminal symbols, as well as appropriate feature-passing to instantiate the top-level features. The only structural difference between the two grammars was a flatter phrase structure for the induced grammar:[5]

```
S   --> NP Vi
        S.tr = NP.f
        S.rel = Vi.h
    --> NP Vc P NP
        S.tr = NP(1).f
        S.lm = NP(2).f
    --> NP Vt NP
        S.tr = NP(1).f
        S.lm = NP(2).f
        S.rel = Vt.h
NP  --> Det N
        NP.f = N.f
```

The lexical productions and feature assignments for N, P, Vi, Vt, Vt, and Det are as in the example grammar.

As for the base SCFG case (discussed in Section 4.5.2), a beam search finds a deeper phrase structure with a somewhat higher posterior probability:

```
S --> NP VP
        S.tr = NP.f
        S.lm = VP.g
        S.rel = VP.h
VP --> Vi
```

[5]As usual we rename the arbitrary nonterminal and feature names generated by the implementation to be close to the target grammar, to make the result more intelligible.

```
        VP.h = Vi.h
    --> X NP
        VP.g = NP.f
        VP.h = X.h
X --> Vc P
        X.h = P.h
   --> Vt
        X.h = Vt.h
```

Notice again the non-traditional (`Vc P`) chunk, which in this case also happens be a good match for the attribute passing rules. The (`Vc P`) allows collapsing both the CFG productions and the feature equations of the VP expansions for copula and transitive sentences.

5.4.2 Syntactic constraints imposed by attributes

It is possible to find cases where the unstructured feature assignments supplied as part of the samples implicitly favor one phrase structure over another, exhibiting a primitive form of semantically induced syntactic bias. Due to the simplicity of the PAG formalism these interactions are not as pervasive as one might think. Also, purely syntactic contingencies tend to interfere and render the semantically preferred alternative dominant regardless of attributes.

Interestingly, the following example of semantic bias is entirely based on the likelihood (data fit) component of the posterior probability, so the phenomenon is not dependent on the bias embodied in the prior distribution.

We start with a set of sample sentences exhibiting three types of sentence phrase structure. (For brevity, we give one sample sentence for each, but the example would also go through if each structure had an associated set of samples instead.)

```
a circle touches a square
a circle is below a square
below a square is a circle
```

Assume that initial merging and chunking has resulted in the traditional NP and PP phrase categories. The baseline grammar for the example is now as follows; the samples corresponding to the rules are listed for convenience.

```
S --> NP Vt NP          (a circle touches a square)
     S.tr = NP(1).f
     S.rel = Vt.g
     S.lm = NP(2).f
S --> NP Vi PP          (a circle is below a square)
     S.tr = NP.f
     S.rel = PP.g
     S.lm = PP.h
S --> PP Vi NP          (below a square is a circle)
     S.tr = NP.f
     S.rel = PP.g
     S.lm = PP.h
```

Now consider the sequence of steps necessary to arrive at the traditional VP phrase structure. First, two chunks are replaced independently: chunk(`Vt NP`) = `VP1` and chunk(`Vi NP`) = `VP2`, resulting in the grammar

```
S --> NP VP1              VP1 --> Vt NP
        S.tr = NP.f               VP1.g = Vt.g
        S.rel = VP1.g             VP1.h = NP.f
        S.lm = VP1.h
S --> NP VP2              VP2 --> Vi PP
        S.tr = NP.f               VP2.g = PP.g
        S.rel = VP2.g             VP2.h = PP.h
        S.lm = VP2.h
```

A final merging step, merge(`VP1`, `VP2`) = `VP`, gives the familiar result:

```
S --> NP VP               VP --> Vt NP
        S.tr = NP.f              VP.g = Vt.g
        S.rel = VP.g             VP.h = NP.f
        S.lm = VP.h       VP --> Vi PP
                                 VP.g = PP.g
                                 VP.h = PP.h
```

While this sequence of steps seems straightforward, it is not the only one possible. The following alternative sequence uses the same types of operators and yields a grammar that is symmetrical, and therefore equally preferable on syntactic grounds alone (assuming no other distributional evidence favors one or the other).

The alternative sequence of steps starts from the baseline grammar by chunking the 'wrong' constituents, namely, chunk(`NP Vt`) = `XP1` and chunk(`PP Vi`) = `XP2`.

```
S --> XP1 NP              XP1 --> NP Vt
        S.tr = XP1.h              XP2.g = Vt.g
        S.rel = XP1.g             XP1.h = NP.f
        S.lm = NP.f
S --> XP2 NP              VP2 --> PP Vi
        S.tr = NP.f               XP2.g = PP.g
        S.rel = X2.g              XP2.h = PP.h
        S.lm = XP2.h
```

As before, we can then merge the new two phrase categories, merge(`XP1`, `XP2`) = `XP`. The resulting SCFG productions have the same number of nonterminals, productions lengths etc. as the original solution, and should therefore receive the same prior probability, unless a prior is used that explicitly favors certain types of phrase structures (e.g., right-branching over left-branching). However, the feature equations obtained are quite different, as in this case the specification does not collapse into three equations as before.

```
S --> XP NP
        S.tr = XP.h      [0.5]
        S.tr = NP.f      [0.5]
        S.rel = XP.g
        S.lm = NP.f      [0.5]
        S.lm = XP.h      [0.5]
```

The result is that probabilities have to be split (the exact probabilities depend on the sample statistics), thereby lowering the grammar likelihood. This alternative syntactic structure is therefore less preferable than the first, but only due to the attached semantic features.

5.5 Limitations and Extensions

The PAG framework as introduced in this chapter has a number of more or less obvious shortcomings, mainly as a result of our desire to keep the probabilistic model simple enough so that various techniques familiar from earlier models could be used (combinations of multinomials with associated priors, Viterbi approximations, simple merging operators, etc.) Below we mention the most important limitations and possible extensions to remedy them.

5.5.1 More expressive feature constraints

Derivation probabilities for PAGs were defined with carefully chosen conditional independence stipulations in order to make them computationally tractable.

- The marginal probability of the context-free aspect of a derivation, $d_S = \sum_{d_F} P(d_S, d_F|M)$, can be computed independent of the feature part of the grammar, simply by using the standard SCFG rules. In particular, the feature specifications cannot rule out a syntactic structure derived with non-zero probability.
- The feature derivation itself can be written as a product of conditional probabilities (of the LHS features given the RHS features). The reason is that the feature dependencies can be put into a consistent total order.[6]

The computational convenienve, however, comes at the price of reduced expressiveness, especially when compare to the way features are used in traditional non-probabilistic grammars. For example, it rules out a natural account of agreement phenomena using relations among RHS features alone:

```
S --> NP VP
     NP.number = VP.number
NP --> Det N
     NP.number = N.number
VP --> V NO
     VP.number = V.number
...
```

where `number` is assigned in the lexical productions for both N and V. Although the feature equation in the first production is highly intuitive, it would effectively require stating a marginal probability for the joint event of value assignments, as opposed to conditional probabilitiy of one value given the other. However, the

[6]Therefore the strict bottom-up feature format could be relaxed, e.g., by using the notion of *L-attributed* feature specifications (Aho *et al.* 1986).

theory of Markov networks (Pearl 1988) tells us that such a marginal probabilities cannot be assigned locally in a consistent fashion.

A partial solution to this particular problem is the imposition of a global total ordering among agreement features, so that the entire system of constraints is again expressible as a product of conditional probabilities. An ordering based on the linguistic notion of *phrase head* might accomplish this: all RHS features could then only depend on RHS features associated with the distinguished head constituent. In any case, the convenient global bottom-up ordering of feature constraints would have to be given up, and the probabilities of the d_S (string) component of a derivation would no longer be independent of the featural aspects of the grammar.

Systems of unordered constraints can still be given a probabilistic interpretation using well-known concepts from statistical physics. Instead of directly defining probabilities for local feature assignments, we could instead define an *energy function* that expresses the 'badness' of an assignment as an arbitrary positive number. The energy function can be defined by local components, namely as a product of local contributions, e.g., one for each rule. The above rule would thereby be translated into a term that gives low energy if and only if `NP.number = VP.number`. The total energy $E(x)$ of a complete feature assignment x is then used to generate probabilities according to the *Boltzmann distribution*

$$P(x) = \frac{e^{-E(x)/T}}{Z} \quad ,$$

where Z is the normalizing constant (integral over the numerator), and T is a parameter (the *temperature*) that controls the 'peakedness' of the distribution.[7]

This formulation is elegant and intuitively appealing (although it obviates the traditional concept of derivation). However, it carries a heavy computational price: Simply obtaining the probability of a given string/feature assignment pair for various alternative grammars generally requires stochastic simulations in order to compute the constant Z. The posterior probabilities of models can also be evaluated using Monte-Carlo techniques (Neal 1993), but the approach seems too inefficient for the generate-and-test search strategies we have explored so far. On the other hand, the formulation also suggests investigating other learning paradigms, such as stochastic optimization via simulated annealing and Boltzmann machine learning (Geman & Geman 1984; Hinton & Sejnowski 1986).

5.5.2 Hierarchical features

In Chapter 4 we saw that the merging algorithm is quite capable of inducing recursive syntactic rules. However, in the PAG formalism there is no matching recursive structure in the feature descriptions: it is constrained to 'flat' feature structures. As a result, even the L_0 semantics of sentences with simple embedded relative clauses

```
a circle is below a square which is to the left of a triangle which ...
```

[7]A $T = \infty$ all configurations are equally probable, regardless of their energy, whereas at $T = 0$ all probability mass is concentrate on the lowest energy configuration.

cannot be adequately described, let alone learned.[8]

The obvious solution to this representational problem are hierarchical feature structures (f-structures) as used by a number of linguistic theories of grammar (Shieber 1986). However, this raises new problems concerning the 'proper' probabilistic formulation. For example, the set of hierarchical feature specifications over a finite set of feature names becomes infinite, raising the question of an appropriate prior distribution over this space. Also, new and more varied merging operators would have to be introduced to match the increased expressiveness of the formalism, leading to new intricacies for the search procedure. Still, pursuing such an extension, maybe not in the full generality of standard f-structures, is a worthwhile subject for future research.

5.5.3 Trade-offs between context-free and feature descriptions

Returning to the issue of generalized feature constraints, there is also a fundamental question on what evidence such 'hidden' feature constraints could be learned. We have seen in Section 4.5.2 that agreement, for example, can be represented and learned based on posterior probabilities and merging operators, but it became clear that context-free productions are an inadequate formalism for these phenomena. Ideally, one would want to move from a description such as

```
S --> NP_sg VP_sg
  --> NP_pl VP_pl
```

to

```
S --> NP VP
     NP.number = VP.number
```

In general, there are other cases where context-free rules and features provide alternative models for the same distributional facts. In such cases a description length criterion should be used to decide which is the better formulation.

Incidentally, nonterminals themselves may be expressed as features (Gazdar *et al.* 1985; Shieber 1986), thereby eliminating the need for separate descriptions mechanisms. This could be the basis of a unified description length metric that allows fair comparisons of alternative modeling solutions.

5.6 Summary

In this chapter we examined a minimal extension of stochastic context-free grammars that incorporates simple probabilistic attributes (or features). We saw how a pair of feature-oriented operators (feature merging and attribution) together with the existing SCFG operators can induce simple grammars in this formalism, and applied the approach to a rudimentary form of semantics found in the L_0 miniature language

[8] A restricted version in which embedding is restricted to one level and the semantics are flattened out in several features can be learned using the operators described in this chapter.

domain. Interestingly, there are cases where even very simple semantics can help disambiguate syntactic phrase structure during learning, based on purely distributional facts reflected in the model likelihoods, and although the semantics have no hierarchical structure of their own.

The addition of the new operators has highlighted the need for more complex heuristics (equivalent to macro operators) in order to efficiently search the space of grammars. We pointed out several severe limitations of the present formulation, which will require substantial extensions in order to account for more interesting linguistic phenomena.

Chapter 6

Efficient parsing with Stochastic Context-free Grammars

6.1 Introduction

So far we have discussed stochastic context-free grammar mainly from the point of view of learning. A much more standard task is the use of a preexisting SCFG for various problems in computation linguistics applications requiring probabilistic processing. In the literature, SCFGs are used for the selection of parses for ambiguous inputs (Fujisaki *et al.* 1991); to guide the rule choice efficiently during parsing (Jones & Eisner 1992a); to compute island probabilities for non-linear parsing (Corazza *et al.* 1991). In speech recognition, probabilistic context-free grammars play a central role in integrating low-level word models with higher-level language models (Ney 1992), as well as in non-finite state acoustic and phonotactic modeling (Lari & Young 1991). In some work, context-free grammars are combined with scoring functions that are not strictly probabilistic (Nakagawa 1987), or they are used with context-sensitive and/or semantic probabilities (Magerman & Marcus 1991; Magerman & Weir 1992; Jones & Eisner 1992a; Briscoe & Carroll 1993).

Although clearly not a perfect model of natural language, stochastic context-free grammars (SCFGs) are superior to non-probabilistic CFGs, with probability theory providing a sound theoretical basis for ranking, pruning, etc. All of the applications listed above involve (or could potentially make use of) one or more of the following standard tasks, compiled by Jelinek & Lafferty (1991).[1]

1. What is the probability that a given string x is generated by a grammar G?
2. What is the single most likely parse (or derivation) for x?
3. What is the probability that x occurs as a prefix of some string generated by G (the *prefix probability* of x)?

[1] Their paper phrases these problem in terms of context-free probabilistic grammars, but they generalize in obvious ways to other classes of models.

4. How should the parameters (e.g., rule probabilities) in G be chosen to maximize the probability over a *training set* of strings?

The incremental model merging algorithm for SCFGs (Chapter 4) requires either (1) or (2) for efficient operation. Traditional grammar parameter estimation is essentially (4), and is typically also used as a post-processing step to model merging (after the grammar structure has been learned). The algorithm described in this chapter can compute solutions to all four of these problems in a single framework, with a number of additional advantages over previously presented isolated solutions. It was originally developed solely as a general and efficient tool and accessory to the model merging algorithm. We then realized that it also solves task (3) in an efficient and elegant fashion, greatly expanding its usefulness, as described below.

Most probabilistic parsers are based on a generalization of bottom-up chart parsing, such as the CYK algorithm. Partial parses are assembled just as in non-probabilistic parsing (modulo possible pruning based on probabilities), while substring probabilities (also known as 'inside' probabilities) can be computed in a straightforward way. Thus, the CYK chart parser underlies the 'standard' solutions to problems (1) and (4) (Baker 1979), as well as (2) (Jelinek 1985). While the Jelinek & Lafferty (1991) solution to problem (3) is not a direct extension of CYK parsing they nevertheless present their algorithm in terms of its similarities to the computation of inside probabilities.

In our algorithm, computations for tasks (1) and (3) proceed incrementally, as the parser scans its input from left to right; in particular, prefix probabilities are available as soon as the prefix has been seen, and are updated incrementally as it is extended. Tasks (2) and (4) require one more (reverse) pass over the parse table constructed from the input.

Incremental, left-to-right computation of prefix probabilities is particularly important since that is a necessary condition for using SCFGs as a replacement for finite-state language models in many applications, such a speech decoding. As pointed out by Jelinek & Lafferty (1991), knowing probabilities $P(x_0 \ldots x_i)$ for arbitrary prefixes $x_0 \ldots x_i$ enables probabilistic prediction of possible follow-words x_{i+1}, as $P(x_{i+1}|x_0 \ldots x_i) = P(x_0 \ldots x_i x_{i+1})/P(x_0 \ldots x_i)$. These conditional probabilities can then be used as word transition probabilities in a Viterbi-style decoder or to incrementally compute the cost function for a stack decoder (Bahl *et al.* 1983).

Another application where prefix probabilities play a central role is the extraction of n-gram probabilities from SCFGs, a problem that is the subject of Chapter 7. Here, too, efficient incremental computation saves time since the work for common prefix strings can be shared.

The key to most of the features of our algorithm is that it is based on the top-down parsing method for non-probabilistic CFGs developed by Earley (1970). Earley's algorithm is appealing because it runs with best-known efficiency on a number of special classes of grammars. In particular, Earley parsing is more efficient than the bottom-up methods in cases where top-down prediction can rule out potential parses of substrings. The worst-case computational expense of the algorithm (either for the complete input, or the incrementally for each new word) is as good as that of the other known specialized algorithms, but can be substantially better on well-known grammar classes.

Earley's parser (and hence ours) also deals with any context-free rule format in a seamless way, without requiring conversions to Chomsky Normal Form (CNF), as is often assumed. Another advantage is that our probabilistic Earley parser has been extended to take advantage of partially bracketed input, and to return partial parses on ungrammatical input. The latter extension removes one of the common objections against top-down, predictive (as opposed to bottom-up) parsing approaches (Magerman & Weir 1992).

6.2 Overview

The remainder of the chapter proceeds as follows. Section 6.3 briefly reviews the workings of an Earley parser without regard to probabilities. Section 6.4 describes how the parser needs to be extended to compute sentence and prefix probabilities. Section 6.5 deals with further modifications for solving the Viterbi and training tasks, for processing partially bracketed inputs, and for finding partial parses. Section 6.7 discusses miscellaneous issues and relates our work to the literature on the subject. In Section 6.8 we summarize and draw some conclusions.

To get an overall idea of probabilistic Earley parsing it should be sufficient to read Sections 6.3, 6.4.2 and 6.4.4. Section 6.4.5 deals with a crucial technicality, and later sections mostly fill in details and add optional features.

We assume the reader is familiar with the basics of context-free grammar theory, such as given in Aho & Ullman (1972:chapter 2). Jelinek *et al.* (1992) provide a tutorial introduction covering the standard algorithms for the four tasks mentioned in the introduction.

Notation The input string is denoted by x. $|x|$ is the length of x. Individual input symbols are identified by indices starting at 0: $x_0, x_1, \ldots, x_{|x|-1}$. The input alphabet is denoted by Σ. Substrings are identified by beginning and end positions $x_{i\ldots j}$. The variables i, j, k are reserved for integers referring to positions in input strings. As in Chapter 4, Latin capital letters X, Y, Z denote nonterminal symbols. Latin lowercase letters $a, b, \ldots$ are used for terminal symbols. Strings of mixed nonterminal and terminal symbols are written using lowercase Greek letters λ, μ, ν. The empty string is denoted by ϵ.

6.3 Earley Parsing

An Earley parser is essentially a generator that builds left-most derivations of strings, using a given set of context-free productions. The parsing functionality arises because the generator keeps track of all possible derivations that are consistent with the input string up to a certain point. As more and more of the input is revealed the set of possible derivations (each of which corresponds to a parse) can either expand as new choices are introduced, or shrink as a result of resolved ambiguities. In describing the parser it is thus appropriate and convenient to use generation terminology.

The parser keeps a set of *states* for each position in the input, describing all pending derivations.[2]

[2]Earley states are also known as *items* in LR parsing, see Aho & Ullman (1972:section 5.2) and Section 6.7.3.

These state sets together form the Earley *chart*. A state is of the form

$$i: \quad {}_kX \rightarrow \lambda.\mu,$$

where X is a nonterminal of the grammar, λ and μ are strings of nonterminals and/or terminals, and i and k are indices into the input string. States are derived from productions in the grammar. The above state is defined relative to a corresponding production

$$X \rightarrow \lambda\mu$$

with the following semantics:

- The current position in the input is i, i.e., $x_0 \dots x_{i-1}$ have been processed so far.[3] The states describing the parser state at position i are collectively called *state set* i. Note that there is one more state set than input symbols: set 0 describes the parser state before any input is processed, while set $|x|$ contains the states after all input symbols have been processed.
- Nonterminal X was expanded starting at position k in the input, i.e. X generates some substring starting at position k.
- The expansion of X proceeded using the production $X \rightarrow \lambda\mu$, and has expanded the right-hand side (RHS) $\lambda\mu$ up to the position indicated by the dot. The dot thus refers to the current position i.

A state with the dot to the right of the entire RHS is called a *complete* state, since it indicates that the left-hand side (LHS) nonterminal has been fully expanded.

Our description of Earley parsing omits an optional feature of Earley states, the lookahead string. Earley's algorithm allows for an adjustable amount of *lookahead* during parsing, in order to process LR(k) grammars deterministically (and obtain the same computational complexity as specialized LR(k) parsers where possible). The addition of lookahead is orthogonal to our extension to probabilistic grammars, so we will not include it here.

The operation of the parser is defined in terms of three operations that consult the current set of states and the current input symbol, and add new states to the chart. This is strongly suggestive of *state transitions* in finite-state models of language, parsing, etc. This analogy will be explored further in the probabilistic formulation later on.

The three types of transitions operate as follows.

Prediction For each state

$$i: \quad {}_kX \rightarrow \lambda.Y\mu,$$

where Y is a nonterminal anywhere in the RHS, and for all rules $Y \rightarrow \nu$ expanding Y, add states

$$i: \quad {}_iY \rightarrow .\nu \quad .$$

[3] This index is implicit in Earley (1970). We include it here for clarity.

A state produced by prediction is called a *predicted state*. Each prediction corresponds to a potential expansion of a nonterminal in a left-most derivation.

Scanning For each state

$$i: \quad {}_kX \rightarrow \lambda.a\mu,$$

where a is a terminal symbol that matches the current input x_i, add the state

$$i+1: \quad {}_kX \rightarrow \lambda a.\mu$$

(move the dot over the current symbol). A state produced by scanning is called a *scanned state*. Scanning ensures that the terminals produced in a derivation match the input string.

Completion For each complete state

$$i: \quad {}_jY \rightarrow \nu.$$

and each state in set j $(j < i)$

$$j: \quad {}_kX \rightarrow \lambda.Y\mu$$

that has Y to the right of the dot, add

$$i: \quad {}_kX \rightarrow \lambda Y.\mu$$

(move the dot over the current nonterminal). A state produced by completion is called a *completed state*.[4] Each completion corresponds to the end of a nonterminal expansion started by a matching prediction step.

One crucial insight into the working of Earley's algorithm is that, although both prediction and completion feed themselves, there are only a finite number of states that can possibly be produced. Therefore recursive prediction and completion have to terminate eventually, and the parser can proceed to the next input via scanning.

To complete the description we need only specify the initial and final states. The parser starts out with

$$0: \quad {}_0 \quad \rightarrow .S,$$

where S is the sentence nonterminal (note the empty left-hand side). After processing the last symbol, the parser verifies that

$$l: \quad {}_0 \quad \rightarrow S.$$

has been produced (among possibly others), where l is the length of the input x. If at any intermediate stage a state set remains empty (because no states from the previous stage permit scanning) the parse can be aborted because an impossible prefix has been detected.

States with empty LHS such as those above are useful in other contexts, as will be shown in Section 6.5.4. We will collectively refer to them as *dummy states*. Dummy states enter the chart only as a result of initialization, as opposed to being derived from grammar productions.

[4]Note the difference between *complete* and *completed* states: Complete states (those with the dot to the right of the entire RHS) are always the result of a completion step, but completion also produces states which are not yet complete.

It is easy to see that Earley parser operations are *correct*, in the sense that each chain of transitions (predictions, scanning steps, completions) corresponds to a possible (partial) derivation. Intuitively, it is also true that a parser that performs these transitions exhaustively is *complete*, i.e., it finds all possible derivations. Formal proofs of these properties are given in the literature, e.g., Aho & Ullman (1972). The relationship between Earley transitions and derivations will be stated more formally in the next section.

The parse trees for sentences can be reconstructed from the chart contents. We will illustrate this in Section 6.5 when discussing Viterbi parses.

Table 6.1 gives an example for Earley parsing, in the form of a trace of transitions as they are performed by our implementation.

Earley's parser can deal with any type of context-free rule format, even with null or ϵ-productions, i.e., those that replace a nonterminal with the empty string. Such productions do however require special attention, and make the algorithm and its description more complicated than otherwise necessary. In the following sections we assume that no null productions have to be dealt with, and then summarize the necessary changes in Section 6.4.7. One might chose to simply preprocess the grammar to eliminate null productions, a process which is also described.

6.4 Probabilistic Earley Parsing

6.4.1 Stochastic context-free grammars

A stochastic context-free grammar (SCFG) extends the standard context-free formalism by adding probabilities to each production:

$$X \to \lambda \quad [p],$$

where the rule probability p is usually written as $P(X \to \lambda)$. This notation to some extent hides the fact that p is a conditional probability, of production $X \to \lambda$ being chosen, given that X is up for expansion. The probabilities of all rules with the same nonterminal X on the LHS must therefore sum to unity. Context-freeness in a probabilistic setting translates into conditional independence of rule choices. As a result, complete derivations have joint probabilities that are simply the products of the rule probabilities involved.

The probabilities of interest mentioned in Section 6.1 can now be defined formally.

Definition 6.1 The following quantities are defined relative to a SCFG G, a nonterminal X, and a string x over the alphabet of G.[5]

a) The *probability of a (partial) derivation* $\nu_1 \Rightarrow \nu_2 \Rightarrow \ldots \nu_k$ is inductively defined by

1) $P(\nu_1) = 1$

2) $P(\nu_1 \Rightarrow \ldots \Rightarrow \nu_k) = P(X \to \lambda) P(\nu_2 \Rightarrow \ldots \Rightarrow \nu_k)$,

[5] This definition is an expanded and generalized version of the one found in Section 4.2.

(a)

S	$\to$	NP VP	Det	$\to$	a
NP	$\to$	Det N	N	$\to$	circle$\vert$square$\vert$triangle
VP	$\to$	VT NP	VT	$\to$	touches
VP	$\to$	VI PP	VI	$\to$	is
PP	$\to$	P NP	P	$\to$	above$\vert$below

(b)

	a	circle	touches	a	square
$_0 \to$ .S	*scanned*	*scanned*	*scanned*	*scanned*	*scanned*
predicted	$_0$Det $\to$ a.	$_1$N $\to$ circle.	$_2$VT $\to$ touches.	$_3$Det $\to$ a.	$_4$N $\to$ triangle.
$_0S \to$ .NP VP	*completed*	*completed*	*completed*	*completed*	*completed*
$_0$NP $\to$ .Det N	$_0$NP $\to$ Det.N	$_0$NP $\to$ Det N.	$_2$VP $\to$ VT.NP	$_3$NP $\to$ Det.N	$_4$NP $\to$ Det N.
$_0$Det $\to$ .a	*predicted*	$_0$S $\to$ NP.VP	*predicted*	*predicted*	$_3$VP $\to$ VT NP.
	$_1$N $\to$ .circle	*predicted*	$_3$NP $\to$ .Det N	$_5$N $\to$ .circle	$_0$S $\to$ NP VP.
	$_1$N $\to$ .square	$_2$VP $\to$ .VT NP	$_3$Det $\to$ .a	$_4$N $\to$ .square	$_0 \to$ S.
	$_1$N $\to$ .triangle	$_2$VP $\to$ .VI PP		$_4$N $\to$ .triangle	
		$_2$VT $\to$ .touches			
		$_2$VI $\to$ .is			
State set 0	1	2	3	4	5

Table 6.1: Example of non-probabilistic Earley-parsing.

(a) Example grammar for a tiny fragment of English. (b) Earley parser processing the sentence *a circle touches a triangle*.

where $\nu_1, \nu_2, \ldots, \nu_k$ are string of terminals and/or nonterminals, $X \to \lambda$ is production of G, and ν_2 is derived from ν_1 by replacing one occurrence of X with λ.

b) The *string probability* $P(X \stackrel{*}{\Rightarrow} x)$ (of x given X) is the sum of the probabilities of all left-most derivations $X \Rightarrow \ldots \Rightarrow x$ producing x from X.[6]

c) The *sentence probability* $P(S \stackrel{*}{\Rightarrow} x)$ (of x given G) is the string probability given the start symbol S of G. By definition, this is also the probability $P(x|G)$ assigned to x by the grammar G.

d) The *prefix probability* $P(S \stackrel{*}{\Rightarrow}_L x)$ (of x given G) is the sum of the probabilities of all sentence strings having x as a prefix,

$$P(S \stackrel{*}{\Rightarrow}_L x) = \sum_{y \in \Sigma^*} P(S \stackrel{*}{\Rightarrow} xy) \quad .$$

(In particular, $P(S \stackrel{*}{\Rightarrow}_L \epsilon) = 1$).

In the following, we assume that the probabilities in a SCFG are *proper* and *consistent* as defined in Booth & Thompson (1973), and that the grammar contains no *useless nonterminals* (ones that can never appear in a derivation). These restrictions ensure that all nonterminals define probability measures over strings, i.e., $P(X \stackrel{*}{\Rightarrow} x)$ is a proper distribution over x for all X. Formal definitions of these conditions are given in Section 6.4.8.

6.4.2 Earley paths and their probabilities

In order to define the probabilities associated with parser operation on a SCFG, we need the concept of a path, or partial derivation, executed by the Earley parser.

Definition 6.2 a) An *(unconstrained) Earley path*, or simply *path*, is a sequence of Earley states linked by prediction, scanning, or completion. For the purpose of this definition, we allow scanning to operate in 'generation mode,' i.e., all states with terminals to the right of the dot can be scanned, not just those matching the input. (For completed states, the predecessor state is defined to be the complete state from the same state set contributing to the completion.)

b) A path is said to be *constrained* by (or *generate*) a string x if in all scanned states the terminals immediately to the left of the dot, in sequence, form the string x.

c) A path is *complete* if the last state on it matches the first, except that the dot has moved to the end of the RHS.

d) We say that a path *starts with* nonterminal X if the first state on it is a predicted state with X on the LHS.

[6] In a *left-most derivation* each step replaces the nonterminal furthest to the left in the partially expanded string. The order of expansion is actually irrelevant for this definition, due to the multiplicative combination of production probabilities. We restrict summation to left-most derivations to avoid counting duplicates, and because left-most derivations will play an important role later.

e) The *length* of a path is defined as the number of *scanned* states on it.

Note that the definition of path length is somewhat counter-intuitive, but is motivated by the fact that only scanned states correspond directly to input symbols. Thus, the length of a path is always the same as the length of the input string it generates.

A constrained path contains a sequence of states from state set 0 derived by repeated prediction (starting with the initial state), followed by a single state from set 1 produced by scanning the first symbol, followed by a sequence of states produced by completion, followed by a sequence of predicted states, followed by a state scanning the second symbol, and so on. The significance of Earley paths is that they are in a one-to-one correspondence with left-most derivations. This will allow us to talk about probabilities of derivations, strings and prefixes in terms of the actions performed by Earley's parser. From now on, we will use 'derivation' to imply a left-most derivation.

Lemma 6.1 a) An Earley parser generates state

$$i: \quad {}_kX \rightarrow \lambda.\mu,$$

if and only if there is a partial derivation

$$S \stackrel{*}{\Rightarrow} x_{0\ldots k-1}X\nu \Rightarrow x_{0\ldots k-1}\lambda\mu\nu \stackrel{*}{\Rightarrow} x_{0\ldots k-1}x_{k\ldots i-1}\mu\nu$$

deriving a prefix $x_{0\ldots x_{i-1}}$ of the input.

b) For each partial derivation from a nonterminal X there is a *unique* Earley path $\mathcal{P}$ starting at X, and vice-versa, such that the sequence of prediction steps on $\mathcal{P}$ corresponds to the productions applied in the derivation, such that $\mathcal{P}$ generates a terminal prefix of the string derived.

(a) is the invariant underlying the correctness and completeness of Earley's algorithm; it can be proved by induction on the length of a derivation (Aho & Ullman 1972:Theorem 4.9). (b) is the slightly stronger statement the mapping between derivations and paths is one-to-one. It follows by verifying that in a left-most derivation each choice for a nonterminal substitution corresponds to exactly one possible prediction step, or else (a) would be violated.

Since we have established that paths correspond to derivations, it is convenient to associate derivation probabilities directly with paths. The uniqueness condition (b) above, which is irrelevant to the correctness of a standard Earley parser, justifies (probabilistic) counting of paths in lieu of derivations.

Definition 6.3 *The probability $P(\mathcal{P})$ of a path $\mathcal{P}$ is the product of all rule probabilities used in the predicted states occurring on $\mathcal{P}$.*

Lemma 6.2 a) For all paths $\mathcal{P}$ starting with a nonterminal X, $P(\mathcal{P})$ gives the probability of the (partial) derivation represented by $\mathcal{P}$. In particular, the string probability $P(X \stackrel{*}{\Rightarrow} x)$ is the sum of the probabilities of all paths starting with X that are complete and constrained by x.

b) The sentence probability $P(S \stackrel{*}{\Rightarrow} x)$ is the sum of the probabilities of all complete paths starting with the initial state, constrained by x.

c) The prefix probability $P(S \stackrel{*}{\Rightarrow}_L x)$ is the sum of the probabilities of all paths $\mathcal{P}$ starting with the initial state, constrained by x, that end in a scanned state.

Note that when summing over all paths "starting with the initial state," summation is actually over all paths starting with S, by definition of the initial state $_0 \rightarrow .S$. (a) follows directly from our definitions of derivation probability, string probability, path probability and the one-to-one correspondence between paths and derivations established by Lemma 6.1. (b) follows from (a) by using S as the start nonterminal. To obtain the prefix probability in (c), we need to sum the probabilities of all complete derivations that generate x as a prefix. The constrained paths ending in scanned states represent exactly the beginnings of all such derivations. Since the grammar is assumed to be consistent and without useless nonterminals, all partial derivations can be completed with probability one. Hence the sum over the constrained incomplete paths is the sought-after sum over all complete derivations generating the prefix.

6.4.3 Forward and inner probabilities

Since string and prefix probabilities are the result of summing derivation probabilities, the goal is to compute these sums efficiently by taking advantage of the Earley control structure. This can be accomplished by attaching two probabilistic quantities to each Earley state, as follows. The terminology is derived from analogous or similar quantities commonly used in the literature on Hidden Markov Models (HMMs) (Rabiner & Juang 1986) and in Baker (1979).

Definition 6.4 The following definitions are relative to an implied input string x.

a) The *forward probability* $\alpha_i({}_kX \rightarrow \lambda.\mu)$ is the sum of the probabilities of all constrained paths of length i that end in state ${}_kX \rightarrow \lambda.\mu$.

b) The *inner probability* $\gamma_i({}_kX \rightarrow \lambda.\mu)$ is the sum of the probabilities of all paths of length $i-k$ that start in state $k : {}_kX \rightarrow .\lambda\mu$ and end in $i : {}_kX \rightarrow \lambda.\mu$, and generate the input symbols $x_k \ldots x_{i-1}$.

It helps to interpret these quantities in terms of an unconstrained Earley parser that operates as a generator emitting—rather than recognizing—strings. Instead of tracking all possible derivations, the generator traces along a single Earley path randomly determined by always choosing among prediction steps according to the associated rule probabilities. Notice that the scanning and completion steps are deterministic once the rules have been chosen.

Intuitively, the forward probability $\alpha_i({}_kX \rightarrow \lambda.\mu)$ is the probability of an Earley generator producing the prefix of the input up to position $i-1$ while passing through state ${}_kX \rightarrow \lambda.\mu$ at position i. However, due to left-recursion in productions the same state may appear several times on a path, and each occurrence is counted towards the total α_i. Thus, α_i is really the *expected number* of occurrences of the given state in state set i. Having said that, we will refer to α simply as a probability, both for the sake of brevity, and to keep the analogy to the HMM terminology of which this is a generalization.[7] Note that for scanned states, α is always a probability, since by definition a scanned state can occur only once along a path.

The inner probabilities, on the other hand, represent the probability of generating a substring of the input from a given nonterminal, using a particular production. Inner probabilities are thus conditional on the presence of a given nonterminal X with expansion starting at position k, unlike the forward probabilities, which include the generation history starting with the initial state. The inner probabilities as defined here correspond closely to the quantities of the same name in Baker (1979). The sum of γ of all states with a given LHS X is exactly Baker's inner probability for X.

The following lemma is essentially a restatement of Lemma 6.2 in terms of forward and inner probabilities. It shows how to obtain the sentence and string probabilities we are interested in, provided that forward and inner probabilities can be computed effectively.

Lemma 6.3 The following assumes an Earley chart constructed by the parser on an input string x with $|x| = l$.

a) Provided that $S \stackrel{*}{\Rightarrow}_L x_{0\ldots k-1}X\nu$ is a possible left-most derivation of the grammar (for some ν), the probability that a nonterminal X generates the substring $x_k \ldots x_{i-1}$ can be computed as the sum

$$P(X \stackrel{*}{\Rightarrow} x_{k\ldots i-1}) = \sum_{i:_k X \to \lambda.} \gamma_i({}_kX \to \lambda.)$$

(sum of inner probabilities over all complete states with LHS X and start index k).

b) In particular, the string probability $P(S \stackrel{*}{\Rightarrow} x)$ can be computed as[8]

$$\begin{aligned} P(S \stackrel{*}{\Rightarrow} x) &= \gamma_l({}_0 \to S.) \\ &= \alpha_l({}_0 \to S.) \end{aligned}$$

c) The prefix probability $P(S \stackrel{*}{\Rightarrow}_L x)$, with $|x| = l$, can be computed as

$$P(S \stackrel{*}{\Rightarrow}_L x) = \sum_{{}_kX \to \lambda x_{l-1}.\mu} \alpha_l({}_kX \to \lambda x_{l-1}.\mu)$$

(sum of forward probabilities over all scanned states).

The restriction in (a) that X be preceded by a possible prefix is necessary since the Earley parser at position i will only pursue derivations that are consistent with the input up to position i. This constitutes the main distinguishing feature of Earley parsing compared to the strict bottom-up computation used in the standard inside probability computation (Baker 1979). There, inside probabilities for all positions and nonterminals are computed, regardless of possible prefixes.

[7] The same technical complication was noticed by Wright (1990) in the computation of probabilistic LR parser tables. The relation to LR parsing will be discussed in Section 6.7.3. Incidentally, a similar interpretation of forward 'probabilities' is required for HMMs with non-emitting states.

[8] The definitions of forward and inner probabilities coincide for the final state.

6.4.4 Computing forward and inner probabilities

Forward and inner probabilities not only subsume the prefix and string probabilities, they are also straightforward to compute during a run of Earley's algorithm. In fact, if it weren't for left-recursive and unit productions their computation would be trivial. For the purpose of exposition we will therefore ignore the technical complications introduced by these productions for a moment, and then return to them once the overall picture has become clear.

During a run of the parser both forward and inner probabilities will be attached to each state, and updated incrementally as new states are created through one of the three types of transition. Both probabilities are set to unity for the initial state $_0 \quad \to .S$. This is consistent with the interpretation that the initial state is derived from a dummy production $\to S$ for which no alternatives exist.

The parse then proceeds as usual, with the probabilistic computations detailed below. The probabilities associated with new states will be computed as sums of various combinations of old probabilities. As new states are generated by prediction, scanning, and completion, certain probabilities have to be *accumulated*, corresponding to the multiple paths leading to a state. That is, if the same state is generated multiple times, the previous probability associated with it has to be *incremented* by the new contribution just computed. States and probability contributions can be generated in any order, as long as the summation for one state is finished before its probability enters into the computation of some successor state. Section 6.6.2 suggests a way to implement this incremental summation.

Notation A few intuitive abbreviations are used from here on to describe Earley transitions succinctly. (1) To avoid unwieldy $\sum$ notation we adopt the following convention. The expression x += y means that x is computed incrementally as a sum of various y terms, which are computed in some order and accumulated to finally yield the value of x.[9] (2) Transitions are denoted by $\Longrightarrow$, with predecessor states on the left and successor states on the right. (3) The forward and inner probabilities of states are notated in brackets after each state, e.g.,

$$i: \quad {}_kX \to \lambda.Y\mu \quad [\alpha, \gamma]$$

is shorthand for $\alpha = \alpha_i({}_kX \to \lambda.Y\mu)$, $\gamma = \gamma_i({}_kX \to \lambda.Y\mu)$.

Prediction (probabilistic)

$$i: \quad {}_kX \to \lambda.Y\mu \quad [\alpha, \gamma] \quad \Longrightarrow \quad i: \quad {}_iY \to .\nu \quad [\alpha', \gamma']$$

for all productions $Y \to \nu$. The new probabilities can be computed as

$$\begin{aligned} \alpha' &\mathrel{+}= \alpha P(Y \to \nu) \\ \gamma' &= P(Y \to \nu) \end{aligned}$$

Note that only the forward probability is accumulated; γ is not used.

[9] This notation suggests a simple implementation, being obviously borrowed from the programming language *C*.

Rationale. α' is the sum of all path probabilities leading up to ${}_kX \to \lambda.Y\mu$, times the probability of choosing production $Y \to \nu$. The value γ' is just a special case of the definition.

Scanning (probabilistic)

$$i: \quad {}_kX \to \lambda.a\mu \quad [\alpha, \gamma] \quad \Longrightarrow \quad i+1: \quad {}_kX \to \lambda a.\mu \quad [\alpha', \gamma']$$

for all states with terminal a matching input at position i. Then

$$\begin{aligned} \alpha' &= \alpha \\ \gamma' &= \gamma \end{aligned}$$

Rationale. Scanning does not involve any new choices since the terminal was already selected as part of the production during prediction.[10]

Completion (probabilistic)

$$\left.\begin{array}{ll} i: & {}_jY \to \nu. \quad [\alpha'', \gamma''] \\ j: & {}_kX \to \lambda.Y\mu \quad [\alpha, \gamma] \end{array}\right\} \quad \Longrightarrow \quad i: \quad {}_kX \to \lambda Y.\mu \quad [\alpha', \gamma']$$

Then

$$\alpha' \mathrel{+}= \alpha\gamma'' \tag{6.1}$$

$$\gamma' \mathrel{+}= \gamma\gamma'' \tag{6.2}$$

Note that α' is not used.

Rationale. To update the old forward/inner probabilities α and γ to α' and γ', respectively, the probabilities of all paths expanding $Y \to \nu$ have to be factored in. These are exactly the paths summarized by the inner probability γ''.

6.4.5 Coping with recursion

The standard Earley algorithm, together with the probability computations described in the previous section would be sufficient *if* it weren't for the problem of recursion in the prediction and completion steps.

The non-probabilistic Earley algorithm can stop recursing as soon as all predictions/completions yield states already contained in the current state set. For the computation of probabilities, however, this would mean truncating the probabilities resulting from the repeated summing of contributions.

[10] In different parsing scenarios the scanning step may well modify probabilities. For example, if the input symbols themselves have attached likelihoods these can be integrated by multiplying them onto α and γ when a symbol is scanned. That way it is possible to perform efficient Earley parsing with integrated joint probability computation directly on weighted lattices of input symbols.

6.4.5.1 Prediction loops

As an example, consider the following simple left-recursive SCFG.

$$\begin{array}{lll} S & \rightarrow a & [p] \\ S & \rightarrow Sb & [q] \quad , \end{array}$$

where $q = 1 - p$. Non-probabilistically, the prediction loop at position 0 would stop after producing the states

$$\begin{array}{lcl} {}_0 & \rightarrow & .S \\ {}_0S & \rightarrow & .a \\ {}_0S & \rightarrow & .Sb \quad . \end{array}$$

This would leave the forward probabilities at

$$\begin{array}{rcl} \alpha_0({}_0S \rightarrow .a) & = & p \\ \alpha_0({}_0S \rightarrow .Sb) & = & q \quad , \end{array}$$

corresponding to just two out of an infinity of possible paths. The correct forward probabilities are obtained as a sum of infinitely many terms, accounting for all possible paths of length 1.

$$\begin{array}{rcl} \alpha_0({}_0S \rightarrow .a) & = & p + qp + q^2p + \ldots = p(1-q)^{-1} = 1 \\ \alpha_0({}_0S \rightarrow .Sb) & = & q + q^2 + q^3 + \ldots = q(1-q)^{-1} = p^{-1}q \end{array}$$

In these sums each p corresponds to a choice of the first production, each q to a choice of the second production. If we didn't care about finite computation the resulting geometric series could be computed by letting the prediction loop (and hence the summation) continue indefinitely.

Fortunately, all repeated prediction steps, including those due to left-recursion in the productions, can be collapsed into a single, modified prediction step, and the corresponding sums computed in closed form. For this purpose we need a probabilistic version of the well-known parsing concept of a *left corner*, which is also at the heart of the prefix probability algorithm of Jelinek & Lafferty (1991).

Definition 6.5 The following definitions are relative to a given SCFG G.

a) Two nonterminals X and Y are said to be in a *left-corner relation* $X \rightarrow_L Y$ iff there exists a production for X that has a RHS starting with Y,

$$X \rightarrow Y\lambda \quad .$$

b) The *probabilistic left-corner relation*[11] $P_L = P_L(G)$ is the matrix of probabilities $P(X \rightarrow_L Y)$, defined as the total probability of choosing a production for X that has Y as a left corner:

$$P(X \rightarrow_L Y) = \sum_{X \rightarrow Y\lambda \in G} P(X \rightarrow Y\lambda) \quad .$$

[11] If a *probabilistic relation* R is replaced by its set-theoretic version R', i.e., $(x, y) \in R'$ iff $R(x, y) \neq 0$, then the closure operations used here reduce to their traditional discrete counterparts; hence the choice of terminology.

c) The relation $X \stackrel{*}{\Rightarrow}_L Y$ is defined as the reflexive, transitive closure of $X \rightarrow_L Y$, i.e., $X \stackrel{*}{\Rightarrow}_L Y$ iff $X = Y$ or there is a nonterminal Z such that $X \rightarrow_L Z$ and $Z \stackrel{*}{\Rightarrow}_L Y$.

d) The *probabilistic reflexive, transitive left-corner relation* $R_L = R_L(G)$ is a matrix of probability sums $R(X \stackrel{*}{\Rightarrow}_L Y)$. Each $R(X \stackrel{*}{\Rightarrow}_L Y)$ is defined as a series

$$\begin{aligned} R(X \stackrel{*}{\Rightarrow}_L Y) &= P(X = Y) \\ &+P(X \rightarrow_L Y) \\ &+\sum_{Z_1} P(X \rightarrow_L Z_1)P(Z_1 \rightarrow_P Y) \\ &+\sum_{Z_1, Z_2} P(X \rightarrow_L Z_1)P(Z_1 \rightarrow_P Z_2)P(Z_2 \rightarrow_P Y) \\ &+\ldots \end{aligned}$$

Alternatively, R_L is defined by the recurrence relation

$$R(X \stackrel{*}{\Rightarrow}_L Y) = \delta(X,Y) + \sum_{Z} P(X \rightarrow_L Z)R(Z \stackrel{*}{\Rightarrow}_L Y)$$

(δ denotes the Kronecker delta, defined as unity if $X = Y$, and zero otherwise).

The recurrence for R_L can be conveniently written in matrix notation

$$R_L = I + P_L R_L,$$

from which the closed-form solution is derived:

$$R_L = (I - P_L)^{-1}.$$

An existence proof for R_L is given in Section 6.4.8. Appendix 6.6.3.1 shows how to speed up the computation of R_L by inverting only a reduced version of the matrix $I - P_L$.

The significance of the matrix R_L for the Earley algorithm is that its elements are the sums of the probabilities of the potentially infinitely many prediction paths leading from a state ${}_kX \rightarrow \lambda.Z\mu$ to a predicted state ${}_iY \rightarrow .\nu$, via any number of intermediate states.

R_L can be computed once for each grammar, and used for table-lookup in the following, modified prediction step.

Prediction (probabilistic, transitive)

$$i: \quad {}_kX \rightarrow \lambda.Z\mu \quad [\alpha, \gamma] \quad \Longrightarrow \quad i: \quad {}_iY \rightarrow .\nu \quad [\alpha', \gamma']$$

for all productions $Y \rightarrow \nu$ such that $R(Z \stackrel{*}{\Rightarrow}_L Y)$ is non-zero. Then

$$\alpha' \mathrel{+}= \alpha R(Z \stackrel{*}{\Rightarrow}_L Y)P(Y \rightarrow \nu) \tag{6.3}$$

$$\gamma' = P(Y \rightarrow \nu) \tag{6.4}$$

Note the new $R(Z \stackrel{*}{\Rightarrow}_L Y)$ factor in the updated forward probability, which accounts for the sum of all path probabilities linking Z to Y. For $Z = Y$ this covers the case of a single step of prediction; $R(Y \stackrel{*}{\Rightarrow}_L Y) \geq 1$ always, since R_L is defined as a reflexive closure.

6.4.5.2 Completion loops

As in prediction, the completion step in the Earley algorithm may imply an infinite summation, and could lead to an infinite loop if computed naively. However, only *unit productions*[12] can give rise to cyclic completions.

The problem is best explained by studying an example. Consider the grammar

$$\begin{array}{lll} S & \rightarrow a & [p] \\ S & \rightarrow T & [q] \\ T & \rightarrow S & [1] \quad , \end{array}$$

where $q = 1 - p$. Presented with the input a (the only string the grammar generates), after one cycle of prediction, the Earley chart contains the following states.

$$\begin{array}{rcll} 0:{}_0 & \rightarrow & .S & \alpha = 1, \quad \gamma = 1 \\ 0:{}_0S & \rightarrow & .T & \alpha = p^{-1}q, \quad \gamma = q \\ 0:{}_0T & \rightarrow & .S & \alpha = p^{-1}q, \quad \gamma = 1 \\ 0:{}_0S & \rightarrow & .a & \alpha = p^{-1}p = 1, \quad \gamma = p. \end{array}$$

The p^{-1} factors are a result of the left-corner sum $1 + q + q^2 + \ldots = (1-q)^{-1}$.

After scanning ${}_0S \rightarrow .a$, completion without truncation would enter an infinite loop. First ${}_0T \rightarrow .S$ is completed, yielding a complete state ${}_0T \rightarrow S.$, which allows ${}_0S \rightarrow .T$ to be completed, leading to another complete state for S, etc. The non-probabilistic Earley parser can just stop here, but as in prediction, this would lead to truncated probabilities. The sum of probabilities that needs to be computed to arrive at the correct result contains infinitely many terms, one for each possible loop through the $T \rightarrow S$ production. Each such loop adds a factor of q to the forward and inner probabilities. The summations for all completed states turn out as

$$\begin{array}{rcll} 1:{}_0S & \rightarrow & x. & \alpha = 1, \gamma = p \\ 1:{}_0T & \rightarrow & S. & \alpha = p^{-1}q(p + pq + pq^2 + \ldots) = p^{-1}q, \gamma = p + pq + pq^2 + \ldots = 1 \\ 1:{}_0 & \rightarrow & S. & \alpha = p + pq + pq^2 + \ldots = 1, \gamma = p + pq + pq^2 + \ldots = 1 \\ 1:{}_0S & \rightarrow & T. & \alpha = p^{-1}q(p + pq + pq^2 + \ldots) = p^{-1}q, \gamma = q(p + pq + pq^2 + \ldots) = q \end{array}$$

The approach taken here to compute exact probabilities in cyclic completions is mostly analogous to that for left-recursive predictions. The main difference is that unit productions, rather than left-corners, form the underlying transitive relation. Before proceeding we can convince ourselves that this is indeed the only case we have to worry about.

[12] Some authors refer to unit productions as *chain productions*.

Lemma 6.4 Let

$$_{k_1}X_1 \to \lambda_1 X_2. \Longrightarrow {}_{k_2}X_2 \to \lambda_2 X_3. \Longrightarrow \ldots \Longrightarrow {}_{k_c}X_c \to \lambda_c X_{c+1}.$$

be a completion cycle, i.e., $k_1 = k_c$, $X_1 = X_c$, $\lambda_1 = \lambda_c$, $X_2 = X_{c+1}$. Then it must be the case that $\lambda_1 = \lambda_2 = \ldots = \lambda_c = \epsilon$, i.e., all productions involved are unit productions $X_1 \to X_2, \ldots, X_c \to X_{c+1}$.

Proof. For all completion chains it is true that the start indices of the states are monotonically increasing, $k_1 \geq k_2 \geq \ldots$ (a state can only complete an expansion that started at the same or a previous position). From $k_1 = k_c$ follows that $k_1 = k_2 = \ldots = k_c$. Because the current position (dot) also refers to the same input index in all states, all nonterminals $X_1, X_2, \ldots, X_c$ have been expanded into the same substring of the input between k_1 and the current position. By assumption the grammar contains no nonterminals that generate ϵ,[13] therefore we must have $\lambda_1 = \lambda_2 = \ldots = \lambda_c = \epsilon$, q.e.d.

We now formally define the relation between nonterminals mediated by unit productions, analogous to the left-corner relation.

Definition 6.6 The following definitions are relative to a given SCFG G.

a) Two nonterminals X and Y are said to be in a *unit-production relation* $X \to Y$ iff there exists a production for X that has Y as its RHS.

b) The *probabilistic unit-production relation* $P_U = P_U(G)$ is the matrix of probabilities $P(X \to Y)$.

c) The relation $X \stackrel{*}{\Rightarrow} Y$ is defined as the reflexive, transitive closure of $X \to Y$, i.e., $X \stackrel{*}{\Rightarrow} Y$ iff $X = Y$ or there is a nonterminal Z such that $X \to Z$ and $Z \stackrel{*}{\Rightarrow} Y$.

d) The *probabilistic reflexive, transitive unit-production relation* $R_U = R_U(G)$ is the matrix of probability sums $R(X \stackrel{*}{\Rightarrow} Y)$. Each $R(X \stackrel{*}{\Rightarrow} Y)$ is defined as a series

$$\begin{aligned} R(X \stackrel{*}{\Rightarrow} Y) &= P(X = Y) \\ &\quad + P(X \to Y) \\ &\quad + \sum_{Z_1} P(X \to Z_1) P(Z_1 \to Y) \\ &\quad + \sum_{Z_1, Z_2} P(X \to Z_1) P(Z_1 \to Z_2) P(Z_2 \to Y) \\ &\quad + \ldots \\ &= \delta(X, Y) + \sum_{Z} P(X \to Z) R(Z \stackrel{*}{\Rightarrow} Y) \quad . \end{aligned}$$

As before, a matrix inversion can compute the relation R_U in closed form:

$$R_U = (I - P_U)^{-1}.$$

[13] Even with null productions, these would not be used for Earley transitions, see Section 6.4.7.

The existence of R_U is shown in Section 6.4.8.

The modified completion loop in the probabilistic Earley parser can now use the R_U matrix to collapse all unit completions into a single step. Note that we still have to do iterative completion on non-unit productions.

Completion (probabilistic, transitive)

$$\left.\begin{array}{ll} i: & {}_jY \rightarrow \nu. \quad [\alpha'', \gamma''] \\ j: & {}_kX \rightarrow \lambda.Z\mu \quad [\alpha, \gamma] \end{array}\right\} \implies \quad i: \quad {}_kX \rightarrow \lambda Z.\mu \quad [\alpha', \gamma']$$

for all Y, Z such that $R(Z \stackrel{*}{\Rightarrow} Y)$ is non-zero, and $Y \rightarrow \nu$ is not a unit production ($|\nu| > 1$). Then

$$\begin{aligned} \alpha' \quad &\mathrel{+}= \quad \alpha\gamma'' R(Z \stackrel{*}{\Rightarrow} Y) \\ \gamma' \quad &\mathrel{+}= \quad \gamma\gamma'' R(Z \stackrel{*}{\Rightarrow} Y) \end{aligned}$$

6.4.6 An example

Consider the grammar

$$\begin{aligned} S \quad &\rightarrow \quad a \quad [p] \\ S \quad &\rightarrow \quad SS \quad [q] \end{aligned}$$

where $q = 1 - p$. This highly ambiguous grammar generates strings of any number of a's, using all possible binary parse trees over the given number of terminals. The states involved in parsing the string aaa are listed in Table 6.2, along with their forward and inner probabilities. The example illustrates how the parser deals with left-recursion and the merging of alternative sub-parses during completion.

Since the grammar has only a single nonterminal, the left-corner matrix P_L has rank 1:

$$P_L = [q] \quad .$$

Its transitive closure is

$$R_L = (I - P_L)^{-1} = [p]^{-1} = [p^{-1}] \quad .$$

Consequently, the example trace shows that the factor p^{-1} being introduced into the forward probability terms in a number of prediction steps.

The sample string can be parsed as either $(a(aa))$ or $((aa)a)$, each parse having a probability of p^3q^2. The total string probability is thus $2p^3q^2$, the computed α and γ values for the final state. The α values for the scanned states in sets 1, 2 and 3 are the prefix probabilities for a, aa, and aaa, respectively: $P(S \stackrel{*}{\Rightarrow}_L a) = 1$, $P(S \stackrel{*}{\Rightarrow}_L aa) = q$, $P(S \stackrel{*}{\Rightarrow}_L aaa) = (1 + p)q^2$.

6.4.7 Null productions

Null productions $X \rightarrow \epsilon$ introduce some complications into the relatively straightforward parser operation described so far, some of which are due specifically to the probabilistic aspects of parsing. This

(a)

$$
\begin{array}{lcll}
S & \rightarrow & a & [p] \\
S & \rightarrow & SS & [q]
\end{array}
$$

(b)

	α	γ
State set 0		
$_0 \rightarrow .S$	1	1
predicted		
$_0S \rightarrow .a$	$1 \cdot p^{-1}p = 1$	p
$_0S \rightarrow .SS$	$1 \cdot p^{-1}q = p^{-1}q$	q
State set 1		
scanned		
$_0S \rightarrow a.$	$p^{-1}p = 1$	p
completed		
$_0S \rightarrow S.S$	$p^{-1}q \cdot p = q$	$q \cdot p = pq$
predicted		
$_1S \rightarrow .a$	$q \cdot p^{-1}p = q$	p
$_1S \rightarrow .SS$	$q \cdot p^{-1}q = p^{-1}q^2$	q
State set 2		
scanned		
$_1S \rightarrow a.$	q	p
completed		
$_1S \rightarrow S.S$	$p^{-1}q^2 \cdot p = q^2$	$q \cdot q = pq$
$_0S \rightarrow SS.$	$q \cdot p = pq$	$pq \cdot p = p^2q$
$_0S \rightarrow S.S$	$p^{-1}q \cdot p^2q = pq^2$	$q \cdot p^2q = p^2q^2$
$_0 \rightarrow S.$	$1 \cdot p^2q = p^2q$	$1 \cdot p^2q = p^2q$
predicted		
$_2S \rightarrow .a$	$(q^2 + pq^2) \cdot p^{-1}p = (1+p)q^2$	p
$_2S \rightarrow .SS$	$(q^2 + pq^2) \cdot p^{-1}q = (1+p^{-1})q^3$	q
State set 3		
scanned		
$_2S \rightarrow a.$	$(1+p)q^2$	p
completed		
$_2S \rightarrow S.S$	$(1+p^{-1})q^3 \cdot p = (1+p)q^3$	$q \cdot p = pq$
$_1S \rightarrow SS.$	$q^2 \cdot p = pq^2$	$pq \cdot p = p^2q$
$_1S \rightarrow S.S$	$p^{-1}q^2 \cdot p^2q = pq^3$	$q \cdot p^2q = p^2q^2$
$_0S \rightarrow SS.$	$pq^2 \cdot p + q \cdot p^2q = 2p^2q^2$	$p^2q^2 \cdot p + pq \cdot p^2q = 2p^3q^2$
$_0S \rightarrow S.S$	$p^{-1}q \cdot 2p^3q^2 = 2p^2q^3$	$q \cdot 2p^3q^2 = 2p^3q^3$
$_0 \rightarrow S.$	$1 \cdot 2p^3q^2 = 2p^3q^2$	$1 \cdot 2p^3q^2 = 2p^3q^2$

Table 6.2: Earley chart as constructed during the parse of aaa.

The grammar is depicted in (a). The two columns to the right in (b) list the forward and inner probabilities, respectively, for each state. In both α and γ columns, the $\cdot$ separates old factors from new ones (as per equations 6.1, 6.2, 6.3). Addition indicates multiple derivations of the same state.

section summarizes the necessary modifications to process null productions correctly, using the previous description as a baseline. Our treatment of null productions follows the (non-probabilistic) formulation of Graham *et al.* (1980), rather than the original one in Earley (1970).

6.4.7.1 Computing ϵ-expansion probabilities

The main problem with null productions is that they allow multiple prediction-completion cycles in between the scanning steps (since null productions do not have to be matched against one or more input symbols). Our strategy will be to collapse all predictions and completions due to chains of null productions into the regular prediction and completion steps, not unlike the way recursive predictions/completions were handled in Section 6.4.5.

A prerequisite for this approach is to precompute, for all nonterminals X, the probability that X expands to the empty string. Note that this is another recursive problem, since X itself may not have a null production, but expand to some nonterminal Y that does.

Computation of $P(X \stackrel{*}{\Rightarrow} \epsilon)$ for all X can be cast as a system of non-linear equations, as follows. For each X, let e_X be an abbreviation for $P(X \stackrel{*}{\Rightarrow} \epsilon)$. By way of example, if X has productions

$$\begin{array}{lll} X & \rightarrow & \epsilon \quad [p_1] \\ & \rightarrow & Y_1 Y_2 \quad [p_2] \\ & \rightarrow & Y_3 Y_4 Y_5 \quad [p_3] \\ & \vdots & \end{array}$$

The semantics of context-free rules imply that X can only expand to ϵ if *all* of the RHSs in one of X's productions expands to ϵ. Translating to probabilities, we obtain the equation

$$e_X = p_1 + p_2 e_{Y_1} e_{Y_2} + p_3 e_{Y_3} e_{Y_4} e_{Y_5} + \ldots$$

In other words, each production contributes a term in which the rule probability is multiplied by the product of the e variables corresponding to the RHS nonterminals, unless the RHS contains a terminal (in which case the production contributes nothing to e_X because it cannot possibly lead to ϵ).

The resulting non-linear system can be solved by iterative approximation. Each variable e_X is initialized to $P(X \rightarrow \epsilon)$, and then repeatedly updated by substituting in the equation right-hand sides, until the desired level of accuracy is attained. Convergence is guaranteed since the e_X values are monotonically increasing and bounded above by the true values $P(X \stackrel{*}{\Rightarrow} \epsilon) \leq 1$. For grammars without cyclic dependencies among ϵ-producing nonterminals this procedure degenerates to simple backward substitution. Obviously the system has to be solved only once for each grammar.

The probability e_X can be seen as the precomputed inner probability of an expansion of X to the empty string, i.e., it sums the probabilities of all Earley paths that derive ϵ from X. This is the justification for the way these probabilities can be used in modified prediction and completion steps, described next.

6.4.7.2 Prediction with null productions

Prediction is mediated by the left-corner relation. For each X occurring to the right of a dot, we generate states for all Y that are reachable from X by way of the $X \rightarrow_L Y$ relation. This reachability criterion has to be extended in the presence of null productions. Specifically, if X has a production $X \rightarrow Y_1 \dots Y_{i-1} Y_i \lambda$ then Y_i is a left corner of X iff $Y_1, \dots, Y_{i-1}$ all have a non-zero probability of expanding to ϵ. The contribution of such a production to the left-corner probability $P(X \rightarrow_L Y_i)$ is

$$P(X \rightarrow Y_1 \dots Y_{i-1} Y_i \lambda) \prod_{k=1}^{i-1} e_{Y_k}$$

The old prediction procedure can now be modified in two steps. First, replace the old P_L relation by the one that takes into account null productions, as sketched above. From the resulting P_L compute the reflexive transitive closure R_L, and use it to generate predictions as before.

Second, when predicting a left corner Y with a production $Y \rightarrow Y_1 \dots Y_{i-1} Y_i \lambda$, add states for all dot positions up to the first RHS nonterminal that cannot expand to ϵ, say from $X \rightarrow .Y_1 \dots Y_{i-1} Y_i \lambda$ through $X \rightarrow Y_1 \dots Y_{i-1} . Y_i \lambda$. We will call this procedure 'spontaneous dot shifting.' It accounts precisely for those derivations that expand the RHS prefix $Y_1 \dots Y_{i-1}$ without consuming any of the input symbols.

The forward and inner probabilities of the states thus created are those of the first state $X \rightarrow .Y_1 \dots Y_{i-1} Y_i \lambda$, multiplied by factors that account for the implied ϵ-expansions. This factor is just the product $\prod_{k=1}^{j} e_{Y_k}$, where j is the dot position.

6.4.7.3 Completion with null productions

Modification of the completion step follows a similar pattern. First, the unit-production relation has to be extended to allow for unit-production chains due to null productions. A rule $X \rightarrow Y_1 \dots Y_{i-1} Y_i Y_{i+1} \dots Y_j$ can effectively act like a unit production that links X and Y_i if all other nonterminals on the RHS can expand to ϵ. Its contribution to the unit production relation $P(X \rightarrow Y_i)$ will then be

$$P(X \rightarrow Y_1 \dots Y_{i-1} Y_i Y_{i+1} \dots Y_j) \prod_{k \neq i} e_{Y_k}$$

From the resulting revised P_U matrix we compute the closure R_U as usual.

The second modification is another instance of spontaneous dot shifting. When completing a state $X \rightarrow \lambda . Y \mu$ and moving the dot to get $X \rightarrow \lambda Y . \mu$, additional states have to be added, obtained by moving the dot further over any nonterminals in μ that have non-zero ϵ-expansion probability. As in prediction, forward and inner probabilities are multiplied by the corresponding ϵ-expansion probabilities.

6.4.7.4 Eliminating null productions

Given these added complications one might consider simply eliminating all ϵ-productions in a preprocessing step. This is mostly straightforward and analogous to the corresponding procedure for non-probabilistic CFGs (Aho & Ullman 1972:Algorithm 2.10). The main difference is the updating of rule probabilities, for which the ϵ-expansion probabilities are again needed.

1. Delete all null productions, except on the start symbol (in case the grammar as a whole produces ϵ with non-zero probability). Scale the remaining production probabilities to sum to unity.
2. For each original rule $X \to \lambda Y \mu$ that contains a nonterminal Y such that $Y \stackrel{*}{\Rightarrow} \epsilon$:
   (a) Create a variant rule $X \to \lambda\mu$
   (b) Set the rule probability of the new rule to $e_Y P(X \to \lambda Y \mu)$. If the rule $X \to \lambda\mu$ already exists, sum the probabilities.
   (c) Decrement the old rule probability by the same amount.

   Iterate these steps for all occurrences of a null-able nonterminal in a RHS.

The crucial step in this procedure is the addition of variants of the original productions that simulate the null productions by deleting the corresponding nonterminals from the RHS. The spontaneous dot shifting described in the previous sections effectively performs the same operation on the fly as the rules are used in prediction and completion.

6.4.8 Existence of R_L and R_U

In Section 6.4.5 we defined the probabilistic left-corner and unit-production matrices R_L and R_U, respectively, to collapse recursions in the prediction and completion steps. It was shown how these matrices could be obtained as the result of matrix inversions. In this appendix we give a proof that the existence of these inverses is assured if the grammar is well-defined in the following three senses. The terminology used here is taken from Booth & Thompson (1973).

Definition 6.7 For a SCFG G over an alphabet Σ, with start symbol S, we say that[14]

a) G is *proper* iff for all nonterminals X the rule probabilities sum to unity, i.e.,

$$\sum_{\lambda:(X\to\lambda)\in G} P(X \to \lambda) = 1 \quad .$$

b) G is *consistent* iff it defines a probability distribution over finite strings, i.e.,

$$\sum_{x\in\Sigma^*} P(S \stackrel{*}{\Rightarrow} x) = 1 \quad ,$$

where $P(S \stackrel{*}{\Rightarrow} x)$ is induced by the rule probabilities according to Definition 6.1(a).

[14] Unfortunately, the terminology used in the literature is not uniform. For example, Jelinek & Lafferty (1991) use the term 'proper' to mean (c), and 'well-defined' for (b). They also state mistakenly that (a) and (c) together are a sufficient condition for (b). Booth & Thompson (1973) show that one can write a SCFG that satisfies (a) and (c) but generates derivations that do not terminate with probability 1, and give necessary and sufficient conditions for (b).

c) G has *no useless nonterminals* iff all nonterminals X appear in at least one derivation of some string $x \in \Sigma^*$ with non-zero probability, i.e., $P(S \stackrel{*}{\Rightarrow} \lambda X \mu \stackrel{*}{\Rightarrow} x) > 0$.

It is useful to translate consistency into 'process-oriented' terms. We can view an SCFG as a stochastic string-rewriting process, in which each step consists of simultaneously replacing all nonterminals in a sentential form with the right-hand sides of productions, randomly drawn according to the rule probabilities. Booth & Thompson (1973) show that the grammar is consistent if and only if the probability D_n that stochastic rewriting of the start symbol S leaves nonterminals remaining after n steps, goes to 0 as $n \rightarrow \infty$. More loosely speaking, rewriting S has to terminate after a finite number of steps with probability 1, or else the grammar is inconsistent.

We observe that the same property holds not only for S, but for all nonterminals, if the grammar has no useless terminals. If any nonterminal X admitted infinite derivations with non-zero probability, then S itself would have such derivations, since by assumption X is reachable from S with non-zero probability.

To prove the existence of R_L and R_U, it is sufficient to show that the corresponding geometric series converge:

$$\begin{aligned} R_L &= I + P_L + P_L^2 + \ldots = (I - P_L)^{-1} \\ R_U &= I + P_U + P_U^2 + \ldots = (I - P_U)^{-1} \quad . \end{aligned}$$

Lemma 6.5 If G is a proper, consistent SCFG without useless nonterminals the powers P_L^n of the left-corner relation, and P_U^n of the unit-production relation converge to zero as $n \rightarrow \infty$.

Proof. Entry (X, Y) in the left-corner matrix P_L is the probability of generating Y as the immediately succeeding left-corner below X. Similarly, entry (X, Y) in the nth power P_L^n is the probability of generating Y as the left-corner of X with $n - 1$ intermediate nonterminals. Certainly $P_L^n(X, Y)$ is bounded above by the probability that the entire derivation starting at X terminates after n steps, since a derivation couldn't terminate without expanding the left-most symbol to a terminal (as opposed to a nonterminal). But that probability tends to 0 as $n \rightarrow \infty$, and hence so must each entry in P_L^n.

For the unit-production matrix P_U a similar argument applies, since the length of a derivation is at least as long as it takes to terminate any initial unit-production chain.

Lemma 6.6 If G is a proper, consistent SCFG without useless nonterminals the series for R_L and R_U as defined above converge to finite, non-negative values.

Proof. P_L^n converging to 0 implies that the magnitude of P_L's largest eigenvalue (its spectral radius) is < 1, which in turn implies that the series $\sum_{i=0}^{\infty} P_L^i$ converges (similarly for P_U).

The elements of R_L and R_U are non-negative since they are the result of adding and multiplying among the non-negative elements of P_L and P_U, respectively.

Interestingly, a SCFG may be inconsistent and still have converging left-corner and/or unit-production matrices, i.e., consistency is a stronger constraint. For example

$$S \rightarrow a \quad [p]$$

$$S \rightarrow SS \quad [q]$$

is inconsistent for any choice of $q \geq \frac{1}{2}$, but the left-corner relation (a single number in this case) is well-defined for all $q < 1$, namely $(1-q)^{-1} = p^{-1}$. Informally speaking, in these cases the left fringe of a derivation is guaranteed to result in a terminal after finitely many steps, but in the remainder of the derivation nonterminals are generated at a faster rate than are they are replaced by terminals.

SCFG consistency will again play an important role in Chapter 7 where the computation of various global quantities such as n-gram expectation, derivation entropy, etc., for a given SCFG will be discussed.

6.4.9 Complexity issues

The probabilistic extension of Earley's parser preserves the original control structure in most aspects, the major exception being the collapsing of cyclic predictions and unit completions, which can only make these steps more efficient. We can therefore apply the complexity analysis from Earley (1970) essentially unchanged. Below we repeat the highlights, together with proof outlines. We also analyze the dependence on the size of the grammar, and compare the result with the other known algorithms for SCFGs.

The key factor in upper-bounding both time and space complexity in Earley's algorithm is the maximal number of states created in each state set, for each input position. An Earley state combines a production, a dot position, and a start index. Productions and dot positions combine to give a number that equals the sum of the lengths of all productions, which is roughly the total 'size' of the grammar. For fully parameterized CNF grammars the number of dotted rules is $O(n^3)$, where n is the number of nonterminals. (A fully parameterized CNF grammar is one in which each triple of nonterminals X, Y, Z forms a production $X \rightarrow YZ$ of non-zero probability.) In both cases the start index contributes a factor of at most l, the length of the input.

6.4.9.1 Scaling with input length

To determine the complexity in terms of l we note that during prediction and scanning, each state is processed exactly once, performing operations that depend only on the size of the grammar, not l; both therefore take $O(l)$. During completion, the worst case is obtained if each of the $O(l)$ states is the result of completing predecessors (with dot positions further left) from all possible previous positions. The total time taken here is thus $O(l^2)$. The completion step thus dominates the computation time, and gives $O(l^3)$ total run time over the entire string.

Earley (1970) identifies several important classes of context-free grammars on which the algorithm runs faster without special modifications. Grammars with no or bounded ambiguity result in completions that have to combine at most a fixed number of previous states (combining completions correspond to coalescing multiple parses for substrings). Such a completion step therefore takes time $O(l)$, to give a total of $O(l^2)$. CFGs that can be processed deterministically, i.e., where the correct choice of rule can be determined using only the history of the parser and a bounded lookahead into the input, such as LR(k) grammars, result in an

Earley chart that contains only a fixed maximum number of states per position. (To realize the benefit of deterministic parsing one generally needs the lookahead feature of Earley's original version, not discussed here.) Prediction, scanning and completion all take constant time (in terms of l) in this case, so the overall time complexity is linear.

If our parser is to be used for on-line computation of prefix probabilities it is critical to know the incremental time complexity for the processing of the next input symbol (or word). From the analysis above we get a worst case incremental time of $O(l^2)$, $O(l)$ for bounded ambiguity grammars, and constant time for deterministic grammars. Since l in this case actually refers to the length of the prefix, incremental processing is generally slowed down as more of the input is incorporated in the chart.

The space complexity in terms of l is $O(l^2)$ since l state sets of $O(l)$ elements each have to be created.

All in all, we get the same time $O(l^3)$, space $O(l^2)$ bounds as in the Inside/Outside (Baker 1979) and LRI (Jelinek & Lafferty 1991) algorithms, with the advantage of better results on known grammar classes.

6.4.9.2 Scaling with grammar size

We will not try to give a precise characterization in the case of sparse grammars (Section 6.6.3 gives some hints on how to implement the algorithm efficiently for such grammars). However, for fully parameterized grammars in CNF we can verify the scaling of our algorithm in terms of the number of nonterminals n, and compare it to the I/O and LRI algorithms, which both run in time $O(n^3)$.

As already mentioned, the number of states per position is $O(ln^3)$ for a CNF grammar. During prediction, summation of forward probabilities (equation 6.3) can be implemented efficiently as follows. We first compute the sum of all α's referring to a given nonterminal X right of the dot, for all X. This can be done in a single pass over the current state set, i.e., in time $O(ln^3)$. The result is a vector of α-sums, indexed by nonterminals. Multiplying this vector with the matrix R_L we get another vector (in time $O(n^2)$). The α' from equation (6.3) are obtained by multiplying the rule probability $P(Y \to \nu)$ with the Y element in that vector (total time for this step $O(n)$).

Scanning involves shifting the dot in the $O(ln)$ states that represent terminal productions. During completion we again have to update probabilities for $O(ln^3)$ states, each of which is the result of summing over $O(l)$ predecessors. (Note that there can be no cyclic completions with CNF grammars.) To implement summations (6.1) and (6.2) efficiently we first sum the inner probabilities γ'' from all states that refer to the same LHS nonterminal in a single $O(n^3)$ pass.

Finally, the matrix inversion to compute the left-corner and unit-production relation matrices is also accomplished in $O(n^3)$ time. However, that cost can be amortized over all subsequent uses of the parser.

The space requirements of all algorithms discussed here are proportional to the number of parameters, i.e., $O(n^3)$.

Overall, we get the same $O(n^3)$ dependence on the number of nonterminals as for the I/O and LRI algorithm.

6.4.10 Summary

To summarize, the modified, probabilistic Earley algorithm works by executing the following steps for each input position.

- Apply a single prediction step to all incomplete states in the current state set. All transitive predictions are subsumed by consulting the left-corner matrix R_L.

  Forward probabilities are computed by multiplying old α's with rule probabilities. Inner probabilities are initialized to their respective rule probabilities.

- A single scanning step applied to all states with terminals to the right of the dot yield the initial elements for the next state set. If the next set remains empty (no scanned states) the parse is aborted.

  Forward and inner probabilities remain unchanged by scanning.

  The sum of all forward probabilities of successfully scanned states gives the current prefix probability.

- Apply iterative completion (highest start index first, breadth-first) to all states, except those corresponding to unit productions. Unit production cycles are subsumed by consulting the matrix R_U.

  Forward and inner probabilities are updated by multiplying old forward and inner probabilities with the inner probabilities of completed expansions.

  The probabilities that nonterminals X generate particular substrings of the input can be computed as sums of inner probabilities $\gamma({}_kX \to \lambda.)$

After processing the entire string in this way, the sentence probability can be read off of either the α or γ of the final state.

6.5 Extensions

This section discusses extensions to the Earley algorithm that go beyond simple parsing and the computation of prefix and string probabilities. These extension are all quite straightforward and well-supported by the original Earley chart structure, which leads us to view them as part of a single, unified algorithm for solving the tasks mentioned in the introduction.

6.5.1 Viterbi parses

Definition 6.8 A *Viterbi parse* for a string x, in a grammar G, is a left-most derivation that assigns maximal probability to x, among all possible derivations for x.

Both the definition of Viterbi parse, and its computation are straightforward generalizations of the corresponding notion for Hidden Markov Models (Rabiner & Juang 1986), where one computes the Viterbi *path* (state sequence) through an HMM. Precisely the same approach can be used in the Earley parser, using the fact that each derivation corresponds to a path.

Path probabilities are recursively multiplied during completion steps using the inner probabilities as accumulators. Summation of probabilities occurs whenever alternative sub-parses lead to a single state. The computation of the Viterbi parse has two parts:

- During the forward pass each state must keep track of the maximal path probability leading to it, as well as the predecessor states associated with that maximum probability path.
- Once the final state is reached, the maximum probability parse can be recovered by tracing back the path of 'best' predecessor states.

The following modifications to the probabilistic Earley parser will implement the forward phase of the Viterbi computation.

- Each state computes an additional probability, its *Viterbi probability* v.
- Viterbi probabilities are propagated in the same way as inner probabilities, *except* that during completion the summation is replaced by maximization: $v_i({}_kX \to \lambda Y.\mu)$ is the maximum of all products $v_i({}_jY \to \nu.)v_j({}_kX \to \lambda.Y\mu)$ that contribute to the completed state ${}_kX \to \lambda Y.\mu$. The same-position predecessor ${}_jY \to \nu.$ associated with the maximum is recorded as the Viterbi path predecessor of ${}_kX \to \lambda Y.\mu$ (the other predecessor state ${}_kX \to \lambda.Y\mu$ can be inferred).
- The completion step uses the original recursion without collapsing of unit production loops. Loops are simply avoided, since they can only lower a path's probability. Collapsing of unit-production completions has to be avoided to maintain a continuous chain of predecessors for later backtracing and parse construction.
- The prediction step does not need to be modified for the Viterbi computation.

Once the final state is reached, a recursive procedure can recover the parse tree associated with the Viterbi parse. This procedure takes an Earley state $i : {}_kX \to \lambda.\mu$ as input and produces the Viterbi parse for the substring between k and i as output. (If the input state is not complete ($\mu \neq \epsilon$), the result will be a partial parse tree with children missing from the root node.)

Viterbi-parse ($i : {}_kX \to \lambda.\mu$)

1. If $\lambda = \epsilon$, return a parse tree with root labeled X and no children.
2. Otherwise, if λ ends in a terminal a, let $\lambda' a = \lambda$, and call this procedure recursively to obtain the parse tree

$$T = \text{Viterbi-parse}(i - 1 : {}_kX \to \lambda'.a\mu)$$

Adjoin a leaf node labeled a as the right-most child to the root or T and return T.

3. Otherwise, if λ ends in a nonterminal Y, let $\lambda' Y = \lambda$. Find the Viterbi predecessor state ${}_jY \rightarrow \nu.$ for the current state. Call this procedure recursively to compute

$$T = \text{Viterbi-parse}(j : {}_kX \rightarrow \lambda'.Y\mu)$$

as well as

$$T' = \text{Viterbi-parse}(i : {}_jY \rightarrow \nu.)$$

Adjoin T' to T as the right-most child at the root, and return T.

6.5.2 Rule probability estimation

The rule probabilities in a SCFG can be iteratively estimated using the EM (Expectation-Maximization) algorithm (Dempster *et al.* 1977). The estimation procedure finds a set of parameters that represent a local maximum of the grammar likelihood function $P(D|G)$, given a sample corpus D. The grammar likelihood is given by the product of the string probabilities,

$$P(D|G) = \prod_{x \in D} P(S \stackrel{*}{\Rightarrow} x) \quad ,$$

i.e., the samples are assumed to be distributed identically and independently.

The two steps of this algorithm can be briefly characterized as follows.

E-step: Compute expectations for how often each grammar rule is used, given the corpus D and the current grammar parameters (rule probabilities).

M-step: Reset the parameters so as to maximize the likelihood relative to the expected rule counts found in the E-step.

This procedure is iterated until the parameter values (as well as the likelihood) converge. It can be shown that each round in the algorithm produces a likelihood that is a least a high as the previous one; the EM algorithm is therefore guaranteed to find at least a local maximum of the likelihood function.

EM is a generalization of the well-known Baum-Welch algorithm for HMM estimation (Baum *et al.* 1970); the original formulation for the case of SCFGs is due to Baker (1979). For SCFGs, the E-step involves computing the expected number of times each production is applied in generating the training corpus. After that, the M-step consists of a simple normalization of these counts to yield the new production probabilities.[15]

In this section we examine the computation of production count expectations required for the E-step. The crucial notion introduced by Baker (1979) for this purpose is the 'outer probability' of a nonterminal, or the joint probability that the nonterminal is generated with a given prefix and suffix of terminals. Essentially the same method can be used in the Earley framework, after extending the definition of outer probabilities to apply to arbitrary Earley states.

[15] Alternatively, a maximum posterior estimate may be generated by combining the expected rule counts with a Dirichlet or other prior on the production probabilities, as discussed in Section 2.5.5.1.

Definition 6.9 Given a string x, $|x| = l$, the *outer probability* $\beta_i({}_kX \rightarrow \lambda.\mu)$ of an Earley state is the sum of the probabilities of all paths that

- start with the initial state,
- generate the prefix $x_0 \ldots x_{k-1}$,
- pass through ${}_kX \rightarrow .\nu\mu$, for some ν,
- generate the suffix $x_i \ldots x_{l-1}$ starting with state ${}_kX \rightarrow \nu.\mu$,
- end in the final state.

Outer probabilities complement inner probabilities in that they refer to precisely to those parts of complete paths generating x not covered by the corresponding inner probability $\gamma_i({}_kX \rightarrow \lambda.\mu)$. A potentially confusing aspect of this definition is that the choice of the production $X \rightarrow \lambda\mu$ is *not* part of the outer probability associated with a state ${}_kX \rightarrow \lambda.\mu$. In fact, the definition makes no reference to the first part λ of the RHS: all states sharing the same k, X and μ will have identical β_i.

Intuitively, $\beta_i({}_kX \rightarrow \lambda.\mu)$ is the probability that an Earley parser operating as a string generator yields the prefix $x_{0\ldots k-1}$ and the suffix $x_{i\ldots l-1}$, while passing through state ${}_kX \rightarrow \lambda.\mu$ at position i (which is independent of λ). As was the case for forward probabilities, β is actually an expectation of the number of such states in set i, as unit production cycles can lead to paths that have more than one state fitting this description. Again, we ignore this technicality in our terminology. The term is motivated by the fact that β reduces to the 'outer probability' of X as defined in Baker (1979) if the dot is in final position.

6.5.2.1 Computing expected production counts

Before going into the details of computing outer probabilities we briefly describe their use in obtaining the expected rule counts needed for the E-step in grammar estimation.

Let $c(X \rightarrow \lambda|x)$ denote the expected number of uses of production $X \rightarrow \lambda$ in the derivation of string x. Alternatively, $c(X \rightarrow \lambda|x)$ is the expected number of times that $X \rightarrow \lambda$ is used for prediction in a complete Earley path generating x. Let $c(X \rightarrow \lambda|\mathcal{P})$ be the number of occurrences of predicted states with production $X \rightarrow \lambda$ along a path $\mathcal{P}$.

$$
\begin{aligned}
c(X \rightarrow \lambda|x) &= \sum_{\mathcal{P} \text{ derives } x} P(\mathcal{P}|S \stackrel{*}{\Rightarrow} x)c(X \rightarrow \lambda|\mathcal{P}) \\
&= \frac{1}{P(S \stackrel{*}{\Rightarrow} x)} \sum_{\mathcal{P} \text{ derives } x} P(\mathcal{P}, S \stackrel{*}{\Rightarrow} x)c(X \rightarrow \lambda|\mathcal{P}) \\
&= \frac{1}{P(S \stackrel{*}{\Rightarrow} x)} \sum_{i:{}_iX \rightarrow .\lambda} P(S \stackrel{*}{\Rightarrow} x_{0\ldots i-1}X\nu \stackrel{*}{\Rightarrow} x) \quad .
\end{aligned}
$$

Summation is over all predicted states using $X \rightarrow \lambda$. $P(S \stackrel{*}{\Rightarrow} x_{0\ldots i-1}X\nu \stackrel{*}{\Rightarrow} x)$ is the sum of the probabilities of all paths passing through $i : {}_iX \rightarrow .\lambda$. Inner and outer probabilities have been defined such that this expected count is obtained precisely as the product of the corresponding of γ_i and β_i. Thus, the expected usage count for a rule can be computed as

$$c(X \to \lambda | x) = \frac{1}{P(S \stackrel{*}{\Rightarrow} x)} \sum_{i:_i X \to .\lambda} \beta_i({}_i X \to .\lambda) \gamma_i({}_i X \to .\lambda) \quad .$$

The sum can be computed after completing both forward and backward passes (or during the backward pass itself) by scanning the chart for predicted states.

6.5.2.2 Computing outer probabilities

The outer probabilities are computed by tracing the complete paths from the final state to the start state, in a single backward pass over the Earley chart. Only completion and scanning steps need to be traced back. Reverse scanning leaves outer probabilities unchanged (similar to inner and forward probabilities in the forward pass), so the only operation of concern is reverse completion.

We describe reverse transitions using the same notation as for their forward counterparts, except that each state is written with its outer and inner probabilities.

Reverse completion

$$i: \quad {}_k X \to \lambda Y.\mu \quad [\beta, \gamma] \quad \Longrightarrow \quad \begin{cases} i: & {}_j Y \to \nu. \quad [\beta'', \gamma''] \\ j: & {}_k X \to \lambda.Y\mu \quad [\beta', \gamma'] \end{cases}$$

for all pairs of states ${}_j Y \to \nu.$ and ${}_k X \to \lambda.Y\mu$ in the chart. Then

$$\begin{aligned} \beta' &\mathrel{+}= \gamma''\beta \\ \beta'' &\mathrel{+}= \gamma'\beta \end{aligned}$$

The inner probability γ is not used.

Rationale. Relative to β', β is missing the probability of expanding Y, which is filled in from γ''. The probability of the surrounding of Y is the probability of the surrounding of X, plus the choice of the rule of production for X and the expansion of the partial LHS λ, which are together given by γ'.

Note that the computation makes use of the inner probabilities computed in the forward pass. The particular way in which γ and β were defined turns out to be convenient here, as no reference to the production probabilities themselves needs to be made in the computation.

As in the forward pass, simple (reverse) completion would not terminate in the presence of cyclic unit productions. A version that collapses all such chains of productions is given below.

Reverse completion (transitive)

$$i: \quad {}_k X \to \lambda Z.\mu \quad [\beta, \gamma] \quad \Longrightarrow \quad \begin{cases} i: & {}_j Y \to \nu. \quad [\beta'', \gamma'] \\ j: & {}_k X \to \lambda.Z\mu \quad [\beta', \gamma'] \end{cases}$$

for all pairs of states $_jY \rightarrow \nu.$ and $_kX \rightarrow \lambda.Z\mu$ in the chart, such that the unit-production relation $R(Z \stackrel{*}{\Rightarrow} Y)$ is non-zero. Then

$$\begin{aligned} \beta' &\mathrel{+}= \gamma''\beta \\ \beta'' &\mathrel{+}= \gamma'\beta R(Z \stackrel{*}{\Rightarrow} Y) \end{aligned}$$

The first summation is carried out once for each state $j : {}_kX \rightarrow \lambda.Z\mu$, whereas the second summation is applied for each choice of Z, but only if $X \rightarrow \lambda Z\mu$ is not a unit production.

Rationale. This increments β'' the equivalent of $R(Z \stackrel{*}{\Rightarrow} Y)$ times, accounting for the infinity of surroundings in which Y can occur if it can be derived through cyclic productions. Note that the computation of β' is unchanged, since γ'' already includes an infinity of cyclically generated subtrees for Y, where appropriate.

6.5.3 Parsing bracketed inputs

The estimation procedure described above (and EM-based estimators in general) are only guaranteed to find locally optimal parameter estimates. Unfortunately, it seems that in the case of unconstrained SCFG estimation local maxima present a very real problem, and make success dependent on chance and initial conditions (Lari & Young 1990). Pereira & Schabes (1992) showed that partially bracketed input samples can alleviate the problem in certain cases. The bracketing information constrains the parse of the inputs, and therefore the parameter estimates, steering it clear from some of the suboptimal solutions that could otherwise be found.

A second advantage of bracketed inputs is that they potentially allow more efficient processing, since the space of potential derivations (or equivalently, Earley paths) is reduced. It is therefore interesting to see how any given parser can incorporate partial bracketing information. This is typically not a big problem, but in the case of Earley's algorithm there is a particularly simple and elegant solution.

Consider again the grammar

$$\begin{aligned} S &\rightarrow a \quad [p] \\ S &\rightarrow SS \quad [q] \end{aligned}$$

A partially bracketed input for this grammar would be $a(aa)a$. The parentheses indicate phrase boundaries that any candidate parse has to be consistent with, e.g., there cannot be a parse that has a constituent spanning the first and second a, or the third and fourth. The supplied bracketing can be nested, of course, and need not be complete, i.e., within a bracketing there are still potentially several ways of parsing a substring.

The Earley parser can deal efficiently with partial bracketing information as follows. A partially bracketed input is processed as usual, left-to-right. When a bracketed portion is encountered, the parser invokes itself recursively on the substring delimited by the pair of parentheses encountered. More precisely:

- The recursive parser instance gets to see only the substring as input.

- Its chart is disjoint from the one used by the parent instance. It cannot use states from the parent chart, except those explicitly passed to it (see below). Conversely, when finished, the parent has access only to those states returned explicitly by the child instance.[16] (The first restriction prevents parsing of constituents that cross the left phrase boundary, while the second restriction prevents a violation of the right phrase boundary.)

- The chart of the child is initialized with all incomplete states from the parent's state set at the start position of the substring.

- The child returns to the parent all (and only) the complete states from its last state set. The parent adds the returned states to the state set at the position immediately following the end of the substring, using it as the input for its own completion procedure.

- Thus the recursive parser invocation and the following completion step replaces the usual prediction-scanning-completion cycle for the entire bracketed substring. After the child returns, the parent continues processing regular input symbols, or other bracketed substrings.

- Needless to say, the child parser instance may itself call on recursive instances to deal with nested bracketings.

This recursion scheme is efficient in that it never explicitly *rejects* a parse that would be inconsistent with the bracketing. Instead it *only considers* those parses that are consistent with the bracketing, while continuing to make use of top-down information like a standard Earley parser.

Processing bracketed strings requires no modification to the computation of probabilities. Probabilities are passed between parent and child as part of states, and processed as before. The recursive control structure simply constrains the set of Earley paths considered by the parser, thereby affecting the probabilities indirectly. For example, ambiguous strings may end up with lower inner probabilities because some derivations are inconsistent with the bracketing.

Only the forward pass is directly affected by the bracketing. Both the Viterbi procedure (Section 6.5.1) and the reverse completion pass (Section 6.5.2) only examine the states already in the chart. They are therefore automatically constrained by the bracketing.

Complexity To assess the complexity benefit of bracketing we can extend the analysis of Section 6.4.9, making use of the recursive structure of the algorithm.

In the standard parsing scheme, the time complexity is $O(l^3)$ for an input of length l. Hence, in the recursive scheme each bracketed substring takes time $O(r^3)$, where r is the number of constituents in the substring (which may be either input symbols or nested constituents). The total number of bracketings in a given input string is $O(l)$. If R is an upper bound on r the total time is therefore $O(lR^3)$.

[16] This does not preclude using a shared chart at the implementation level, of course, as long as the above protocol is adhered to.

In a *fully bracketed* input string each grammar rule used in the derivation is reflected in a corresponding bracketing. Hence, r is bounded by the maximal production length of the grammar, and the time complexity is simply $O(l)$.

6.5.4 Robust parsing

In many applications ungrammatical input has to be dealt with in some way. Traditionally it was seen as a drawback of top-down parsing algorithms such as Earley's that they sacrifice 'robustness,' i.e., the ability to find partial parses in an ungrammatical input, for the efficiency gained from top-down prediction (Magerman & Weir 1992).

One approach to the problem is to build robustness into the grammar itself. In the simplest case one could add top-level productions

$$\begin{aligned} S &\rightarrow XS \\ &\rightarrow \epsilon \end{aligned}$$

where X can expand to any nonterminal, including an 'unknown word' category. This grammar will cause the Earley parser to find all partial parses of substrings, effectively behaving like a bottom-up parser constructing the chart in left-to-right fashion. More refined variations are possible: the top-level productions could be used to model which phrasal categories (sentence fragments) can likely follow each other. This probabilistic information can then be used in a pruning version of the Earley parser (Section 6.7.2) to effect a compromise between robust and expectation-driven parsing.

An alternative method for making Earley parsing more robust is to modify the parser itself so as to accept arbitrary input and find all or a chosen subset of possible substring parses. Below we present such a simple extension to Earley's algorithm (probabilistic or not). In the probabilistic version, it will also produce the likelihoods of those partial parses. The potential advantage over the grammar modifying approach is that it can be modified to make use of various criteria for which partial parses to allow at runtime.

The extension for robust parsing does not require any changes to the way the Earley parser operates on the chart, only that the chart be 'seeded' with some extra states before starting. The computation performed as a result of this modification will be essentially equivalent to that of a CYK bottom-up parser, but with the advantage that a single parsing engine can be used for both standard and robust parsing.

6.5.4.1 Seeding the chart

In standard Earley parsing the parser expects to find exactly one instance of an S nonterminal generating the entire input. This expectation is reflected by the fact that the chart is initialized with the dummy start state

$$0 : {}_0 \quad \rightarrow .S \quad .$$

For robust parsing, we want to identify all nonterminals that can possibly generate any substring of the input. This can be accomplished by also placing dummy states

$$k : {}_k \quad \to .X \quad ,$$

for all positions k and nonterminals X, in the Earley chart prior to the start of normal operation. (In practice, dummy states need to be added only for those nonterminals X whose expansion can start with the current input symbol. This technique is discussed in Section 6.6.3.2.)

The immediate effect of these extra states is that more predictions will be generated, from which more completions follow, etc. After finishing the processing of the jth state set, the chart will contain states

$$j : {}_k \quad \to X.$$

indicating that nonterminal X generates the substring $x_{k \ldots j-1}$.

Table 6.3(a) illustrates the robust parsing process using the example grammar from Table 6.1 (p. 128).

Probabilities in the extra states are handled as follows. The initial dummy states ${}_k \to .X$ are initialized with a forward probability of zero. This will ensure that the forward probabilities of all extra states remain at zero and don't interfere with the computation of prefix probabilities from the regular Earley states.

Inner probabilities on dummy states are initialized to unity just as for the S start state, and processed in the usual way. The inner probabilities for the each substring/nonterminal pair can then be read off of the complete dummy states.

Viterbi probabilities and Viterbi back-pointers can also be processed unchanged. Applying the **Viterbi-parse** procedure from Section 6.5.1 to the complete dummy states yields Viterbi parses for all substring/nonterminal pairs.

6.5.4.2 Assembling partial parses

Instead of consulting the chart for individual substring/nonterminal pairs it may be useful to obtain a list of all complete partial parses of the input. A *complete partial parse* is a sequence of nonterminals that together generate the input. For example, using the grammar in Table 6.1, the input *a circle touches above a square* has the complete partial parses 'NP VT PP' and 'Det N VT P NP', among others. The input is grammatical exactly if S is among the complete partial parses.

First note that there may not exist a complete partial parse if the input contains unknown symbols. As a preprocessing step, or on-line during parsing, one may have to create new preterminals to account for such new input symbols.

The Earley algorithm can be minimally extended to also generate the list of all partial parses. What is needed is some device that assembles abutting nonterminals from partial parses left-to-right. This work can be carried out as a by-product of the normal completion process using the concept of a *variable state*. A variable state is a special kind of dummy state in which the RHS can have any number of nonterminals to the

(a)

```
STATE SET 0
            0       --> .S
 predicted ...
            0 S     --> .NP VP
            0 NP    --> .DET N
STATE SET 1
 scanned "a" ...
            0 DET   --> a .
 completed ...
            0       --> DET .
       MAX  0 DET   --> a .
            0 NP    --> DET .N
 predicted ...
STATE SET 2
 scanned "circle" ...
            1 N     --> circle .
 completed ...
            0       --> NP .
            1       --> N .
       MAX  0 NP    --> DET N .
            1 N     --> circle .
            0 S     --> NP .VP
 predicted ...
            2 VP    --> .VI PP
            2 VP    --> .VT NP
STATE SET 3
 scanned "touches" ...
            2 VT    --> touches .
 completed ...
            2       --> VT .
       MAX  2 VT    --> touches .
            2 VP    --> VT .NP
 predicted ...
            3 PP    --> .P NP
STATE SET 4
 scanned "below" ...
            3 P     --> below .
 completed ...
            3       --> P .
       MAX  3 P     --> below .
            3 PP    --> P .NP
 predicted ...
            4 S     --> .NP VP
            4 NP    --> .DET N
STATE SET 5
 scanned "a" ...
            4 DET   --> a .
 completed ...
            4       --> DET .
       MAX  4 DET   --> a .
            4 NP    --> DET .N
 predicted ...
STATE SET 6
 scanned "square" ...
            5 N     --> square .
 completed ...
            3       --> PP .
            4       --> NP .
       MAX  3 PP    --> P NP .
            5       --> N .
            4 NP    --> DET N .
            5 N     --> square .
            4 S     --> NP .VP
```

(b)

```
STATE SET 0
            0       --> .?
STATE SET 1
            0       --> DET .?
STATE SET 2
            0       --> NP .?
            0       --> DET N .?
STATE SET 3
            0       --> NP VT .?
            0       --> DET N VT .?
STATE SET 4
            0       --> NP VT P .?
            0       --> DET N VT P .?
STATE SET 5
            0       --> NP VT P DET .?
            0       --> DET N VT P DET .?
STATE SET 6
            0       --> NP VT PP .?
            0       --> DET N VT PP .?
            0       --> NP VT P NP .?
            0       --> DET N VT P NP .?
            0       --> NP VT P DET N .?
            0       --> DET N VT P DET N .?
```

Table 6.3: Robust parsing using the simple grammar from Table 6.1.

(a) State sets generated from parsing the ungrammatical string *a circle touches above a square*. Dummy states (those with empty LHS) represent partial parses. States representing ‘maximal’ partial parses are marked with MAX. Predictions that don’t lead to completions have been omitted to save space. (b) Trace of variable state completions resulting in a list of complete partial parses for this input.

left of the dot, and a *variable* to the right of the dot, written as a question mark:

$$i : {}_k \quad \rightarrow \lambda.?$$

As usual, such a state means that the nonterminals in λ have generated the substring $x_{k\ldots i-1}$. The variable indicates that any continuation of the nonterminal sequence is allowed.

The variable semantics are taken into account during prediction and completion. A variable state generates predictions for all nonterminals, thus having the same effect as the nonterminal-specific dummy states in the previous section. During completion, a variable state combines with all complete states to yield new variable states:

$$\left.\begin{array}{ll} i : & {}_jY \rightarrow \mu. \\ j : & {}_k \quad \rightarrow \lambda.? \end{array}\right\} \Longrightarrow \quad i : \quad {}_k \quad \rightarrow \lambda Y.?$$

for all Y. (That is, the complete nonterminal Y is inserted before the dot and the variable retained following the dot to allow further completions.) This is implemented by a trivial modification to the standard completion step. Inner probabilities, Viterbi probabilities and Viterbi back-pointers are processed as usual.

The net effect of processing variable states is that all complete partial parses can be read off the final state set in the chart as the right-hand sides of variable states (after discarding the variable itself). The inner probability of a variable state reflects the combined likelihood of the partial parse for the given input. The Viterbi probability of a variable state is the joint maximum achieved by the most likely parses for each of the substrings. Different derivations from the same complete partial parse may split the input string at different places. The **Viterbi-parse** procedure when applied to a variable state will recover the most likely such split.

Table 6.3(b) shows a trace of variable state completions used in enumerating the partial parses for the example given earlier.

The total number of complete partial parses can be exponential in the length of the input. It may therefore be desirable to compute only a subset of them, applying some application-specific filter criterion. One such criterion is that one is only interested in 'maximal' complete partial parses. A complete partial parse is called *maximal* if it has no subsequence of nonterminals that can be replaced by another nonterminal so as to yield another complete partial parse. For example, in the case of *a circle touches above a square*, the only maximal parse is 'NP VT PP'.

It turns out that a filter for maximal partial parses is easy to implement in the Earley framework. Maximal parses contain only nonterminals that are not themselves part of a larger partial parse. Therefore, during completion, we can mark all states that contributed to a larger constituent, and later identify the unmarked states as the ones corresponding to maximal parses. (The chart in Table 6.3(a) has all maximal states labeled with `MAX`.) When completing variable states we simply skip all completions due to non-maximal states. The list of complete partial parses obtained from the chart will then contain precisely the maximal ones.

6.6 Implementation Issues

This section briefly discusses some of the experience gained from implementing the probabilistic Earley parser. Implementation is mainly straightforward and many of the standard techniques for context-free grammars can be used (Graham *et al.* 1980). However, some aspects are unique due to the addition of probabilities.

6.6.1 Prediction

Due to the collapsing of transitive predictions, this step can be implemented in a very efficient and straightforward manner. As explained in Section 6.4.5, one has to perform a single pass over the current state set, identifying all nonterminals Z occurring to the right of dots, and add states corresponding to all productions $Y \rightarrow \nu$ that are reachable through the left-corner relation $Z \stackrel{*}{\Rightarrow}_L Y$. As indicated in equation (6.3), contributions to the forward probabilities of new states have to be summed when several paths lead to the same state. However, the summation in equation (6.3) can be mostly eliminated if the α values for all old states with the same nonterminal Z are summed first, and then multiplied by $R(Z \stackrel{*}{\Rightarrow}_L Y)$. These quantities are then summed over all nonterminals Z, and the result is once multiplied by the rule probability $P(Y \rightarrow \nu)$ to give the forward probability for the predicted state.

6.6.2 Completion

Unlike prediction, the completion step still involves iteration. Each complete state derived by completion can potentially feed other completions. An important detail here is that to ensure that all contributions to a state's α and γ are summed before proceeding with using that state as input to further completion steps.

One approach to this problem is to insert complete states into a prioritized queue. The queue orders states by their start indices, highest first. This is because states corresponding to later expansion always have to be completed first before they can lead to the completion of earlier expansions. For each start index, the entries are managed as a first-in-first-out queue, ensuring that the directed dependency graph formed by the states is traversed in breadth-first order.

A completion pass can now be implemented as follows. Initially, all complete states from the previous scanning step are inserted in the queue. States are then removed from the front of the queue, and used to complete other states. Among the new states thus produced, complete ones are again added to the queue. The process iterates until no more states remain in the queue. Because the computation of probabilities already includes chains of unit productions, states derived from such productions need not be queued, which also ensures that the iteration terminates.

A similar queuing scheme, with the start index order reversed, can be used for the reverse completion step needed in the computation of outer probabilities (Section 6.5.2).

6.6.3 Efficient parsing with large sparse grammars

During work with a moderate-sized, application-specific natural language grammar taken from the BeRP system (Jurafsky *et al.* 1994b) we had opportunity to optimize our implementation of the algorithm. Below we relate some of the lessons learned in the process.

6.6.3.1 Speeding up matrix inversions

Both prediction and completion steps make use of a matrix R defined as a geometric series derived from a matrix P,

$$R = I + P + P^2 + \ldots = (I - P)^{-1} \quad .$$

Both P and R are indexed by the nonterminals in the grammar. The matrix P is derived from the SCFG rules and probabilities (either the left-corner relation or the unit-production relation).

For an application using a fixed grammar the time taken by the precomputation of left-corner and unit-production matrices may not be crucial, since it occurs off-line. There are cases, however, when that cost should be minimal, e.g., when rule probabilities are iteratively reestimated.

Even if the matrix P is sparse, the matrix inversion can be prohibitive for large numbers of nonterminals n. Empirically, matrices of rank n with a bounded number p of non-zero entries in each row (i.e., p is independent of n) can be inverted in time $O(n^2)$, whereas a full matrix of size $n \times n$ would require time $O(n^3)$.

In many cases the grammar has a relatively small number of nonterminals that have productions involving other nonterminals in a left-corner (or the RHS of a unit-production). Only those nonterminals can have non-zero contributions to the higher powers of the matrix P. This fact can be used to substantially reduce the cost of the matrix inversion needed to compute R.

Let P' be a subset of the entries of P, namely, only those elements indexed by nonterminals that have a non-empty row in P. For example, for the left-corner computation, P' is obtained from P by deleting all rows and columns indexed by nonterminals that do not have productions starting with nonterminals. Let I' be the identity matrix over the same set of nonterminals as P'. Then R can be computed as

$$\begin{aligned}
R &= I + (I + P + P^2 + \ldots)P \\
&= I + (I' + P' + P'^2 + \ldots) \star P \\
&= I + (I' - P')^{-1} \star P \\
&= I + R' \star P \quad .
\end{aligned}$$

Here R' is the inverse of $I' - P'$, and $\star$ denotes a matrix multiplication in which the left operand is first augmented with zero elements to match the dimensions of the right operand, P.

The speedups obtained with this technique can be substantial. For a grammar with 789 nonterminals, of which only 132 have nonterminal productions, the left-corner matrix was computed in 12 seconds (including the final multiply with P and addition of I). Inversion of the full matrix $I - P$ took 4 minutes 28 seconds.[17]

[17] These figures are not very meaningful for their absolute values. All measurements were obtained on a Sun SPARCstation 2 with a

6.6.3.2 Efficient prediction

As discussed in Section 6.4.9, the worst-case run-time on fully parameterized CNF grammars is dominated by the completion step. However, this is not necessarily true of sparse grammars. Our experiments showed that the computation is dominated by the generation of Earley states during the prediction steps.

It is therefore worthwhile to minimize the total number of predicted states generated by the parser. Since predicted states only affect the derivation if they lead to subsequent scanning we can use the next input symbol to constrain the relevant predictions. To this end, we compute the *extended left-corner relation* R_{LT}, indicating which *terminals* can appear as left corners of which nonterminals. R_{TL} is a Boolean matrix with rows indexed by nonterminals and columns indexed by terminals. It can be computed as the product

$$R_{LT} = R_L P_{LT}$$

where P_{LT} has a non-zero entry at i, j iff there is a production for nonterminal i that starts with terminals j. R_L is the old left-corner relation.

During the prediction step we can ignore incoming states whose RHS nonterminal following the dot cannot have the current input as a left-corner, and then eliminate from the remaining predictions all those whose LHS cannot produce the current input as a left-corner. These filtering steps are very fast as they involve only table lookup.

On a test corpus this technique cut the number of generated predictions to almost 1/4 and speeded up parsing by a factor of 3.3. The corpus consisted of 1143 sentence with an average length of 4.65 words. The top-down prediction alone generated 991781 states and parsed at a rate of 590 milliseconds per sentence. With bottom-up filtered prediction only 262287 states were generated, resulting in 180 milliseconds per sentence.

A trivial optimization often found in Earley parsers is to precompute the entire first prediction step, as it doesn't depend on the input and may eliminate a substantial portion of the total predictions per sentence.[18] We found that with bottom-up filtering this technique lost its edge: scanning the precomputed predicted states turned out to be slower than computing the zeroth state set filtered by the first input.

6.7 Discussion

6.7.1 Relation to finite-state models

Throughout the exposition of the Earley algorithm and its probabilistic extension we have been alluding, in concepts and terminology, to the algorithms used with probabilistic finite-state models, in particular Hidden Markov Models (Rabiner & Juang 1986). Many concepts carry over, if suitably generalized, most notably that of forward probabilities. Prefix probabilities can be computed from forward probabilities by the Earley parser just as in HMMs because Earley states summarize past history in much the same way as the states in a finite-state model. There are important differences, however. The number of states in an HMM

CommonLisp/CLOS implementation of generic sparse matrices that was not particularly optimized for this task.

[18]The first prediction step accounted for roughly 30% of all predictions on our test corpus.

remains fixed, whereas the number of possible Earley states grows linearly with the length of the input (due to the start index).

Incidentally, the HMM concept of *backward probabilities* has no useful analog in Earley parsing. It is tempting to define $\beta_i(s)$ as the conditional probability that the generator produces the remaining string given that it is currently in state s. Alas, this would be an ill-defined quantity since the generation of a suffix depends (via completion) on more than just the current state.

The solution found by Baker (1979), adopted here in modified form, is to use outer probabilities instead of 'backward' probabilities. Outer probabilities follow the hierarchical structure of a derivation, rather than the sequential structure imposed by left-to-right processing. Fortunately, outer probability computation is just as well supported by the Earley chart as forward and inner probabilities.[19]

6.7.2 Online pruning

In finite-state parsing (especially speech decoding) one often makes use of the forward probabilities for *pruning* partial parses before having seen the entire input. Pruning is formally straightforward in Earley parsers: in each state set, rank states according to their α values, then remove those states with small probabilities compared to the current best candidate, or simply those whose rank exceed a given limit. Notice this will not only omit certain parses, but will also underestimate the forward and inner probabilities of the derivations that remain. Pruning procedures have to be evaluated empirically since they invariably sacrifice completeness and, in the case of the Viterbi algorithm, optimality of the result.

While Earley-based on-line pruning awaits further study, there is reason to believe the Earley framework has inherent advantages over strategies based only on bottom-up information (including so-called 'over-the-top' parsers). Context-free forward probabilities include *all* available probabilistic information (subject to assumptions implicit in the SCFG formalism) available from an input prefix, whereas the usual inside probabilities do not take in account the nonterminal prior probabilities that result from the top-down relation to the start state. Using top-down constraints does not necessarily mean sacrificing robustness, as discussed in Section 6.5.4. On the contrary, by using Earley-style parsing with a set of carefully designed and estimated 'fault tolerant' top-level productions, it should be possible to use probabilities to better advantage in robust parsing. This approach is a subject of ongoing work in the tight-coupling framework of the BeRP system (Jurafsky *et al.* 1994b:see below).

6.7.3 Relation to probabilistic LR parsing

One of the major alternative context-free parsing paradigms besides Earley's algorithm is *LR parsing* (Aho & Ullman 1972). A comparison of the two approaches, both in their probabilistic and non-probabilistic aspects, is interesting and provides useful insights. The following remarks assume familiarity with both approaches. We sketch the fundamental relations, as well as the the important tradeoffs between the two

[19] The closest thing to a HMM backward probability is probably the *suffix probability* $P(S \overset{*}{\Rightarrow}_R x)$.

frameworks.[20]

Like an Earley parser, LR parsing uses dotted productions, or *items*, to keep track of the progress of derivations as the input is processed. (The start indices are not part of LR items: we may therefore use the term *item* to refer to both LR items and Earley states without start indices.) An Earley parser constructs sets of possible items on the fly, by following all possible partial derivations. An LR parser, on the other hand, has access to a complete list of *sets of possible items* computed beforehand, and at runtime simply follows transitions between these sets. The item sets are known as the 'states' of the LR parser.[21] A grammar is suitable for LR parsing if these transitions can be performed deterministically by considering only the next input and the contents of a shift-reduce stack. *Generalized LR parsing* is an extension that allows parallel tracking of multiple state transitions and stack actions by using a graph-structured stack (Tomita 1986).

Probabilistic LR parsing (Wright 1990) is based on LR items augmented with certain conditional probabilities. Specifically, the probability p associated with an LR item $X \rightarrow \lambda.\mu$ is, in our terminology, a normalized forward probability:

$$p = \frac{\alpha_i(X \rightarrow \lambda.\mu)}{P(S \stackrel{*}{\Rightarrow}_L x_{0\ldots i-1})} \quad ,$$

where the denominator is the probability of the current prefix.[22] LR item probabilities, are thus conditioned forward probabilities, and can be used to compute conditional probabilities of next words: $P(x_i|x_{0\ldots i-1})$ is the sum of the p's of all items having x_i to the right of the dot (extra work is required if the item corresponds to a 'reduce' state, i.e., if the dot is in final position).

Notice that the definition of p is independent of i as well as the start index of the corresponding Earley state. Therefore, to ensure that item probabilities are correct independent of input position, item sets would have to be constructed so that their probabilities are unique within each set. However, this may be impossible given that the probabilities can take on infinitely many values and in general depend on the history of the parse. The solution used by Wright (1990) is to collapse items whose probabilities are within a small tolerance ϵ and are otherwise identical. The same threshold is used to simplify a number of other technical problems, e.g., left-corner probabilities are computed by iterated prediction, until the resulting changes in probabilities are smaller than ϵ. Subject to these approximations, then, a probabilistic LR parser can compute prefix probabilities by multiplying successive conditional probabilities for the words it sees.[23]

As an alternative to the computation of LR transition probabilities from a given SCFG, one might instead estimate such probabilities directly from traces of parses on a training corpus. Due to the imprecise relationship between LR probabilities and SCFG probabilities it is not entirely clear if the model thus estimated corresponds to any particular SCFG in the usual sense. However, Briscoe & Carroll (1993) turn

[20]Like Earley parsers, LR parsers can be built using various amounts of *lookahead* to make the operation of the parser (more) deterministic, and hence more efficient. Only the case of zero-lookahead, LR(0), is considered here; the correspondence between LR(k) parsers and k-lookahead Earley parsers is discussed in the literature (Earley 1970; Aho & Ullman 1972).

[21]Once more, it is helpful to compare this to a closely related finite-state concept: the states of the LR parser correspond to sets of Earley states, similar to the way the states of a deterministic FSA correspond to sets of states of an equivalent non-deterministic FSA under the standard subset construction.

[22]The identity of this expression with the item probabilities of Wright (1990) can be proved by induction on the steps performed to compute the p's, as shown in Appendix 6.9. Unfortunately, Wright presents this computation without giving a precise definition of what these numbers are probabilities of.

[23]It is not clear what the numerical properties of this approximation are, e.g., how the errors will accumulate over longer parses.

this incongruity into an advantage by using the LR parser as a probabilistic model its own right, and show how LR probabilities can be extended to capture non-context-free contingencies.

The problem of capturing more complex distributional constraints in natural language is clearly important, but well beyond the scope of this chapter. We simply remark that it should be possible to define 'interesting' non-standard probabilities in terms of Earley parser actions so as to better model non-context-free phenomena. Apart from such considerations, the choice between LR methods and Earley parsing is a typical space-time tradeoff. Even though an Earley parser runs with the same linear time and space complexity as an LR parser on grammars of the appropriate LR class, the constant factors involved will be much in favor of the LR parser as almost all the work has already been compiled into its transition and action table. However, the size of LR parser tables can be exponential in the size of the grammar (due to the number of potential item subsets). Furthermore, if the generalized LR method is used for dealing with non-deterministic grammars (Tomita 1986) the runtime on arbitrary inputs may also grow exponentially.

The bottom line is that each application's needs have to be evaluated against the pros and cons of both approaches to find the best solution. From a theoretical point of view, the Earley approach has the inherent appeal of being the more general (and exact) solution to the computation of the various SCFG probabilities.

6.7.4 Other related work

The literature on Earley-based probabilistic parsers is sparse, presumably because of the precedent set by the Inside/Outside algorithm, which is more naturally formulated as a bottom-up algorithm.

Schabes (1991) shows that Earley's algorithm can be augmented with the computation of inner and outer probabilities, in much the same way as presented here. However, the algorithm presented is not fully general as it is restricted to sentences with bounded ambiguity, i.e., there are no provisions for handling unit production cycles. Schabes focuses on grammar estimation (using generalized inside and outside probabilities) and only mentions the potential for computing prefix probabilities.

Magerman & Marcus (1991) are interested primarily in scoring functions to guide a parser efficiently to the most promising parses. They use Earley-style top-down prediction only to suggest worthwhile parses, not to compute precise probabilities.[24]

Both Nakagawa (1987) and Päseler (1988) use a non-probabilistic Earley parser augmented with 'word match' scoring. Though not truly probabilistic, these algorithms are similar to the Viterbi version described here, in that they find a parse that optimizes the accumulated matching scores (without regard to rule probabilities). Prediction and completion loops do not come into play since no precise inner or forward probabilities are computed.

Dan Jurafsky (personal communication) had written an Earley parser for the Berkeley Restaurant Project (BeRP) speech understanding system that computed forward probabilities for restricted grammars (without left-corner or unit production recursion). The parser now uses the methods described here to provide

[24] They argue that precise probabilities are inappropriate in natural language parsing.

exact SCFG prefix and next-word probabilities to a tightly-coupled speech decoder (Jurafsky *et al.* 1994b).

An essential idea in the probabilistic formulation of Earley's algorithm is the collapsing of recursive predictions and unit completion chains, replacing them by lookups in precomputed matrices. This idea arises in our formulation out of the need to compute probability sums given as infinite series. Graham *et al.* (1980) use a non-probabilistic version of the same technique to create a highly optimized Earley-like parser for general CFGs that implements prediction and completion by operations on Boolean matrices.[25]

The matrix inversion method for dealing with left-recursive prediction is borrowed from the LRI algorithm of Jelinek & Lafferty (1991) for computing prefix probabilities for SCFGs in CNF.[26] We then use that idea a second time to deal with the similar recursion arising from unit productions in the completion step. We suspect, but have not proved, that the Earley computation of forward probabilities when applied to a CNF grammar performs a computation that is in some sense isomorphic to that of the LRI algorithm. In any case, we believe that the parser-oriented view afforded by the Earley framework makes for a more intuitive solution to the prefix probability problem, with the added advantage that it is not restricted to CNF grammars.

Kupiec (1992a) has proposed a version of the Inside/Outside algorithm that allows it to operate on non-CNF grammars. Interestingly, Kupiec's algorithm is also based on a generalization of finite-state models, namely, Recursive Transition Networks (RTNs). Probabilistic RTNs are essentially HMMs that allow nonterminals as output symbols. Also, the dotted productions appearing in Earley states are exactly equivalent to the states in an RTN derived from a CFG.

6.7.5 A simple typology of SCFG algorithms

The various known algorithms for probabilistic CFGs share many similarities, and vary along similar dimensions. One such dimension is whether the quantities entered into the parser chart are defined in a bottom-up (CYK) fashion, or whether left-to-right constraints are an inherent part of their definition.[27]

Another point of variation is the 'sparseness' trade-off. If we are given a set of nonterminals and wanted to list all possible CFG rules involving those nonterminals, the list would be infinite due to the arbitrary length of the right-hand sides of productions. This is a problem, for example, when training a CFG starting with complete ignorance about the structure of the rules.

A workaround is to restrict the rule format somehow, usually to CNF, and then list all possible productions. Algorithms that assume CNF are usually formulated in terms of such a fully parameterized grammar where all triples X, Y, Z form a possible rule $X \rightarrow YZ$ with non-zero probability, although in many cases they may be specialized to handle sparse grammars efficiently.

At the other extreme we have algorithms with accept unrestricted CFG productions and are therefore meant for sparse grammars, where almost all (in the set theoretic sense) possible productions have probability

[25] This connection to the GHR algorithm was pointed out by Fernando Pereira. Exploration of this link then lead to the extension of our algorithm to handle ϵ-productions, as described in Section 6.4.7.

[26] Their method uses the transitive (but not reflexive) closure over the left-corner relation P_L, for which they chose the symbol Q_L. We chose the symbol R_L in this chapter to point to this difference.

[27] Of course a CYK-style parser can operate left-to-right, right-to-left, or otherwise by reordering the computation of chart entries.

	Full CNF	Sparse CFG
Bottom-up	Inside/outside	Stochastic RTNs
	(Baker 1979)	(Kupiec 1992a)
Left-to-right	LRI	Probabilistic
	(Jelinek & Lafferty 1991)	Earley

Table 6.4: Tentative typology of SCFG algorithms according to prevailing directionality and sparseness of the CFG.

zero. It appears that these algorithms tend to be more naturally formulated in terms of a stochastic process, as opposed to static specifications of string probabilities.

To illustrate these points, the algorithms discussed in this section have been arranged in the grid depicted in Table 6.4.

6.8 Summary

We have presented an Earley-based parser for stochastic context-free grammars that is appealing for its combination of advantages over existing methods. Earley's control structure makes it run with best-known complexity on a number of special grammar classes, and no worse than standard bottom-up probabilistic chart parsers on fully parameterized SCFGs.

Unlike bottom-up parsers it also computes accurate prefix probabilities incrementally while scanning its input, along with the usual substring (inside) probabilities. The chart constructed during parsing supports both Viterbi parse extraction and Baum-Welch type rule probability estimation by way of a backward pass over the parser chart. If the input comes with (partial) bracketing to indicate phrase structure this information can be easily incorporated to restrict the allowable parses. A simple extension of the Earley chart allows finding partial parses of ungrammatical input.

The computation of probabilities is conceptually simple, and follows directly Earley's parsing framework, while drawing heavily on the analogy to finite-state language models. It does not require rewriting the grammar into normal form. Thus, the present algorithm fills a gap in the existing array of algorithms for SCFGs, efficiently combining the functionalities and advantages of several previous approaches.

6.9 Appendix: LR item probabilities as conditional forward probabilities

In Section 6.7.3 an interpretation of LR item probabilities as defined in Wright (1990:Section 2.1) was given in terms of the forward probabilities used by the Earley parser. Below we give a proof for the correctness of this interpretation. Notice that these are the 'ideal' LR probabilities that *should* be attached to items, if it weren't for the identification of items with close probabilities to keep the LR state list finite.

Let $p(X \to \nu)$ be the probability for LR item $X \to \nu$ (with a dot somewhere in the RHS). We want to show that

$$p(X \to \lambda.\mu) = \frac{\alpha_i({}_kX \to \lambda.\mu)}{P(S \stackrel{*}{\Rightarrow}_L x_{0\ldots i-1})} \quad , \tag{6.5}$$

for any item $X \to \lambda.\mu$, regardless of position i and start index k. Note that i is not always equal to the position of the last input symbol processed; a reduce action of the parser effectively resets i to the beginning of the reduced nonterminal.

The computation of LR item sets begins with the initial item $\to .S$, which has $p = 1$ by definition, thereby agreeing with (6.5).

The first operation for constructing item sets is *closure*, whereby for each item $X \to \lambda.Y\mu$, all items $Y \to .\nu$ corresponding to the available productions $Y \to \nu$ are added to the set. This operation is recursive and corresponds obviously to Earley's prediction step. Also, the way in which p values are propagated follows exactly the way forward probabilities are handled during prediction. (The left-corner relation R_L could be used to compute closure probabilities exactly, but Wright suggests using a truncated recursion instead.) Since closure and prediction are thus isomorphic, and since the prefix relative to the items does not change, (6.5) also remains valid during this step.

Finally, a successor set I' of kernel items is constructed from an existing closed set I in what corresponds to Earley's scanning or completion. Specifically, for each current item $X \to \lambda.Y\mu \in I$, an item $X \to \lambda Y.\mu$ is placed in I', reachable by scanning a terminal Y or reducing (completing) a nonterminal Y. (We let Y stand for either terminal or nonterminal to treat both cases jointly.) The new item probability is computed as

$$p(X \to \lambda Y.\mu) = \frac{p(X \to \lambda.Y\mu)}{\displaystyle\sum_{Z \to \pi.Y\rho \in I} p(Z \to \pi.Y\rho)} \tag{6.6}$$

This can be understood as scaling the total probability of items matching Y to unity.

By substituting (6.5) into (6.6) we get

$$\begin{aligned}
p(X \to \lambda Y.\mu) &= \frac{\frac{\alpha_i(X \to \lambda.Y\mu)}{P(S \stackrel{*}{\Rightarrow}_L x_{0\ldots i-1})}}{\displaystyle\sum_{Z \to \pi.Y\rho \in I} \frac{\alpha_i(Z \to \pi.Y\rho)}{P(S \stackrel{*}{\Rightarrow}_L x_{0\ldots i-1})}} \\
&= \frac{\alpha_i(X \to \lambda.Y\mu)}{\displaystyle\sum_{Z \to \pi.Y\rho \in I} \alpha_i(Z \to \pi.Y\rho)} \\
&= \frac{\alpha_i(X \to \lambda.Y\mu)\gamma_{i'}(Y)}{\displaystyle\sum_{Z \to \pi.Y\rho \in I} \alpha_i(Z \to \pi.Y\rho)\gamma_{i'}(Y)} \\
&= \frac{\alpha_{i'}(X \to \lambda Y.\mu)}{\displaystyle\sum_{Z \to \pi Y.\rho \in I'} \alpha_{i'}(Z \to \pi Y.\rho)} && (6.7) \\
&= \frac{\alpha_{i'}(X \to \lambda Y.\mu)}{P(S \stackrel{*}{\Rightarrow}_L x_{0\ldots i'-1})} && (6.8)
\end{aligned}$$

$$= \quad p(X \rightarrow \lambda Y.\mu)$$

The position i' is that of the current next input. We have used the abbreviation $\gamma_{i'}(Y)$ for the sum of inner probabilities pertaining to the completed Y, i.e.,

$$\gamma_i(Y) = \begin{cases} 1 & \text{if } Y \text{ is terminal} \\ \sum_{Y \rightarrow \nu.} \gamma_i(Y \rightarrow \nu.) & \text{if } Y \text{ is nonterminal.} \end{cases}$$

Two steps in the derivation above need justification. In (6.7) we are computing forward probabilities just as in an Earley completion step (see equation (6.1)). To get (6.8) we observe that the set I' contains *all* possible kernel items after having processed the prefix $x_{0\ldots i'-1}$ (by definition of the LR parsing method). Hence the sum of $\alpha_{i'}$ represents all possible partial derivations generating the prefix, i.e., $P(S \stackrel{*}{\Rightarrow}_L x_{0\ldots i'-1})$.

Chapter 7

N-grams from Stochastic Context-free Grammars

7.1 Introduction

In Chapter 2 we introduced n-gram models as the one of the simplest, but nevertheless very popular types of probabilistic grammars. Particularly the special cases of bigram and trigram models have proven extremely useful for such tasks as automated speech recognition, part-of-speech tagging, and word-sense disambiguation. However, their obvious linguistic deficiencies also cause a number of practical difficulties in these and other applications. The lack of linguistically motivated structure entails a large number of freely adjustable parameters, which is an illustration of the structure-parameter tradeoff discussed in Section 2.4. As a result, very large corpora are needed for reliable estimation of n-gram models, often requiring additional sophisticated smoothing techniques to avoid the well-known problems of maximum-likelihood estimators (Church & Gale 1991). The lack of linguistic motivation also makes n-gram practically incomprehensible to humans, and impossible to extend and maintain except by brute-force reestimation.

While stochastic context-free grammars (SCFGs) have their own problems, these are to some degree complementary to those of n-grams. SCFGs have many fewer parameters (so can be reasonably trained with smaller corpora), and they capture linguistic generalizations, and are easily understood, written and extended by linguists. For example, if a new word with a number of known possible syntactic categories is to be added it is usually straightforward to do so by adding lexical productions to the SCFG. The structure of the grammar then entails co-occurrence statistics with other words.

This chapter describes a technique for computing an n-gram grammar from an existing SCFG—an attempt to get the best of both worlds. In Section 7.2 we illustrate the basic problem to be solved and provide a number of motivations for its solution. Section 7.3 describes the mathematics underlying the computation, while Section 7.4 addresses questions of complexity and efficient implementation. Section 7.5 discusses the issue of when the n-gram distribution derived from a SCFG is well-defined.

Section 7.6 reports an experiment demonstrating one of the applications of the n-gram-from-SCFG computation, as well as its practical feasibility. Section 7.7 summarizes the algorithm and some general points. In Section 7.8 we briefly touch upon miscellaneous problems that have solutions with close formal ties to the n-gram algorithm.

7.2 Background and Motivation

As defined in Section 2.2.2, an n-gram grammar is a set of probabilities $\mathrm{P}(w_n | w_1 w_2 \ldots w_{n-1})$, giving the probability that w_n follows a word string $w_1 w_2 \ldots w_{n-1}$, for each possible combination of the w's in the vocabulary Σ of the language. For a 5000 word vocabulary, a bigram grammar would have approximately $5000 \times 5000 = 25,000,000$ free parameters, and a trigram grammar would have $\approx 125,000,000,000$.

Even an example of much smaller scale illustrates the problem well. Consider the following simple SCFG (rule probabilities given in brackets):

$$
\begin{array}{lcll}
S & \rightarrow & NP\ VP & [1.0] \\
NP & \rightarrow & N & [0.4] \\
NP & \rightarrow & Det\ N & [0.6] \\
VP & \rightarrow & V & [0.8] \\
VP & \rightarrow & V\ NP & [0.2] \\
Det & \rightarrow & \text{the} & [0.4] \\
Det & \rightarrow & \text{a} & [0.6] \\
N & \rightarrow & \text{book} & [1.0] \\
V & \rightarrow & \text{close} & [0.3] \\
V & \rightarrow & \text{open} & [0.7]
\end{array}
$$

The language generated by this grammar contains 5 words. Including markers for sentence beginning and end, a bigram grammar would contain 6×6 probabilities, or $6 \times 5 = 30$ free parameters (since probabilities must sum to one). A trigram grammar would come with $(5 \times 6 + 1) \times 5 = 155$ parameters. Yet, the above SCFG has only 10 probabilities, only 4 of which are free parameters. The standard deviation of the maximum likelihood estimator for multinomial parameters (equation 2.8) is $O(\frac{1}{\sqrt{c}})$, where c is the number of samples. We therefore expect the estimates for the SCFG parameters to be roughly $\sqrt{30/4} \approx 2.7$ times more reliable than those for the bigram model.[1]

The reason for this discrepancy, of course, is that the *structure* of the SCFG itself is a discrete (hyper-)parameter with considerable potential variation, and has been fixed beforehand. The point is that such a structure is comprehensible by humans, and can in many cases be constrained using prior knowledge, thereby reducing the estimation problem for the remaining probabilities. The problem of estimating SCFG parameters from data is solved with standard techniques, usually by likelihood maximization using the EM

[1] This is under the simplifying assumption that all bigrams and SCFG productions are exercised equally. This is clearly not true; one source of systematic error is that the productions higher up in a SCFG get used more that those closer to the terminals.

algorithm (cf. Sections 2.3.2, 4.2.2, 6.5.2). In the absence of human expertise the grammar induction methods of Chapter 4 or Chapter 3 may be used.[2]

There are good arguments that SCFGs are in principle not adequate probabilistic models for natural languages, due to the conditional independence assumptions they embody (Magerman & Marcus 1991; Jones & Eisner 1992b; Briscoe & Carroll 1993). The main criticisms are that production probabilities are independent of expansion context (e.g., whether a noun phrase is realized in subject of object position), and that lexical co-occurrences, as well as lexical/syntactical contingencies cannot easily be represented, resulting in poor probabilistic estimates for these phenomena. Such shortcomings can be partly remedied by using SCFGs with very specific, semantically oriented categories and rules (Jurafsky *et al.* 1994b). If the goal is to use n-grams nevertheless, then their computation from a more constrained SCFG is still useful since the results can be interpolated with raw n-gram estimates for smoothing. An experiment illustrating this approach is reported below.

On the other hand, even if more sophisticated language models give better results, n-grams will most likely still be important in applications such as speech recognition. The standard speech decoding technique of frame-synchronous dynamic programming (Ney 1984) is based on a first-order Markov assumption, which is satisfied by bigrams models (as well as by Hidden Markov Models), but not by more complex models incorporating non-local or higher-order constraints (including SCFGs). A standard approach is therefore to use simple language models to generate a preliminary set of candidate hypotheses. These hypotheses, e.g., represented as word lattices or N-best lists (Schwartz & Chow 1990), are re-evaluated later using additional criteria that can afford to be more costly due to the more constrained outcomes. In this type of setting, the techniques developed here can be used to compile probabilistic knowledge encoded in the more elaborate language models into n-gram estimates that improve the quality of the hypotheses generated by the decoder.

Finally, comparing directly estimated, reliable n-grams with those compiled from other language models is a potentially useful method for evaluating the models in question.

For the purpose of this chapter we assume that computing n-grams from SCFGs is of either practical or theoretical interest and concentrate on the computational aspects of the problem.

It should be noted that there are alternative, unrelated methods for addressing the problem of the large parameter space in n-gram models. For example, Brown *et al.* (1992) describe an approach based on grouping words into classes, thereby reducing the number of conditional probabilities in the model. Dagan *et al.* (1994) explore similarities between words to interpolate bigram estimates involving words with similar syntagmatic distributions.

The technique of compiling higher-level grammatical models into lower-level is not entirely new: Zue *et al.* (1991) report building a word-pair grammar from more elaborate language models to achieve good coverage, by random generation of sentences. We essentially propose a solution for extending this approach to the probabilistic realm. The need for obtaining n-gram estimates from SCFGs originated in the BeRP speech understanding system already mentioned elsewhere in this thesis (Jurafsky *et al.* 1994a).

[2]This chapter describes an n-gram algorithm specifically for SCFGs. However, the methods described here are easily adapted to the simpler HMM case.

The previous solution to the problem was to estimate n-gram probabilities from the SCFG by counting on randomly generated artificial samples.

7.3 The Algorithm

7.3.1 Normal form for SCFGs

Unlike in other parts of this thesis, we cannot get around the need to *normalize* the grammar to *Chomsky Normal Form (CNF)*. A CFG is in CNF if all productions are of the form

$$X \rightarrow Y\ Z$$

or

$$X \rightarrow a$$

where $X, Y, Z \in \mathcal{N}$ and $a \in \Sigma$.

Any CFG structure can be converted into a weakly equivalent CNF grammar (Hopcroft & Ullman 1979), and in the case of SCFGs the probabilities can be assigned such that the string probabilities remain unchanged.[3] Furthermore, parses in the original grammar can be reconstructed from corresponding CNF parses.

In short, we can, without loss of generality, assume that the SCFGs in question is in CNF. The algorithm described here in fact generalizes to the more general Canonical Two-Form (Graham *et al.* 1980) format, and in the case of bigrams ($n = 2$) it can even be modified to work directly for arbitrary SCFGs. Still, the CNF form is convenient, and to keep the exposition simple we assume all SCFGs to be in CNF.

7.3.2 Probabilities from expectations

The first key insight towards a solution is that the n-gram probabilities can be obtained from the associated *expected frequencies* for n-grams and $(n-1)$-grams:

$$P(w_n|w_1 w_2 \ldots w_{n-1}) = \frac{c(w_1 \ldots w_n|L)}{c(w_1 \ldots w_{n-1}|L)} \tag{7.1}$$

where $c(w|L)$ stands for the expected count of occurrences of the substring w in a sentence of L.[4]

Proof. Write the expectation for n-grams recursively in terms of those of order $n-1$ and the conditional n-gram probabilities:

$$c(w_1 \ldots w_n|L) = c(w_1 \ldots w_{n-1}|L) P(w_n|w_1 w_2 \ldots w_{n-1}).$$

Therefore, if we can compute $c(w|G)$ for all substrings w of lengths n and $n-1$ for a SCFG G, we immediately have an n-gram grammar for the language generated by G.

[3] Preservation of string probabilities is trivial if the grammar has no null or unit productions. In cases where it does, an algorithm similar to the one in Section 6.4.7 can be used to update the probabilities.

[4] The only counts appearing here are expectations, so be will not be using special notation to make a distinction between observed and expected values.

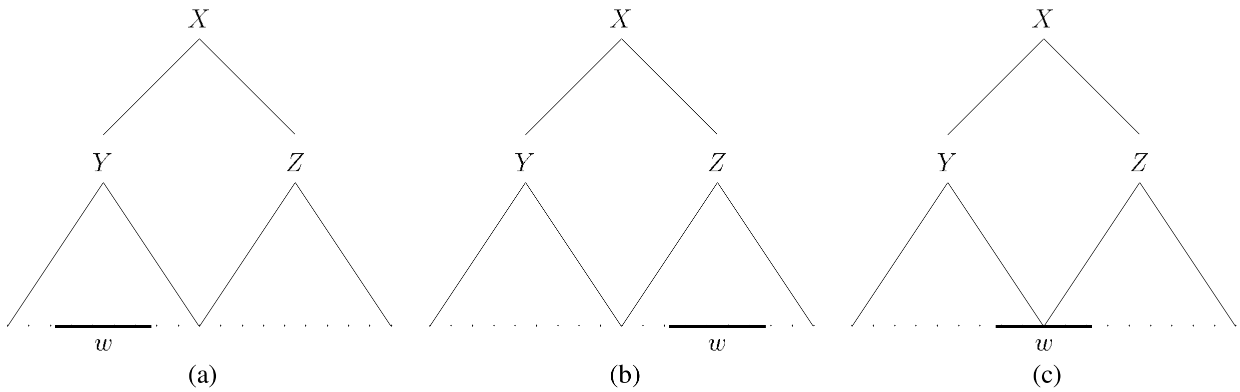


Figure 7.1: Three ways of generating a substring w from a nonterminal X.

Notation Extending the notation used in previous chapters, $X \stackrel{*}{\Rightarrow}_R \alpha$ denotes that non-terminal X generates the string α as a suffix, while $X \stackrel{*}{\Rightarrow}_L \alpha$ means that X generates α as a prefix. $P(X \stackrel{*}{\Rightarrow}_L \alpha)$ and $P(X \stackrel{*}{\Rightarrow}_R \alpha)$ are the probabilities associated with these events.

7.3.3 Computing expectations

Our goal now is to compute the substring expectations for a given grammar. Formalisms such as SCFGs which have a recursive rule structure suggest a divide-and-conquer algorithm that follows the recursive structure of the grammar.

We generalize the problem by considering $c(w|X)$, the expected number of (possibly overlapping) occurrences of $w = w_1 \ldots w_n$ in strings generated by an arbitrary nonterminal X. The special case $c(w|S)$ is the solution sought, where S is the start symbol of the grammar.

Now consider all possible ways that nonterminal X can generate string $w = w_1 \ldots w_n$ as a substring, denoted by $X \stackrel{*}{\Rightarrow} \ldots w_1 \ldots w_n \ldots$, and the associated probabilities. For each production of X we have to distinguish two main cases, assuming the grammar is in CNF. If the string in question is of length 1, $w = w_1$, and if X happens to have a production $X \rightarrow w_1$, then that production adds exactly $P(X \rightarrow w_1)$ to the expectation $c(w|X)$.

If X has non-terminal productions, say, $X \rightarrow YZ$, then w might also be generated by recursive expansion of the right-hand side. Here, for each production, there are three subcases.

(a) First, Y can by itself generate the complete w (see Figure 7.1(a)).

(b) Likewise, Z itself can generate w (Figure 7.1(b)).

(c) Finally, Y could generate $w_1 \ldots w_j$ as a suffix ($Y \stackrel{*}{\Rightarrow}_R w_1 \ldots w_j$) and Z, $w_{j+1} \ldots w_n$ as a prefix ($Z \stackrel{*}{\Rightarrow}_L w_{j+1} \ldots w_n$), thereby resulting in a single occurrence of w (Figure 7.1(c)).

Each of these cases will have an expectation for generating $w_1 \ldots w_n$ as a substring, and the total expectation $c(w|X)$ will be the sum of these partial expectations. The total expectations for the first two cases (that of the substring being completely generated by Y or Z) are given recursively: $c(w|Y)$ and $c(w|Y)$ respectively. The expectation for the third case is

$$\sum_{j=1}^{n-1} P(Y \stackrel{*}{\Rightarrow}_R w_1 \dots w_j) P(Z \stackrel{*}{\Rightarrow}_L w_{j+1} \dots w_n), \tag{7.2}$$

where one has to sum over all possible split points j of the string w.

To compute the total expectation $c(w|X)$, then, we have to sum over all these choices: the production used (weighted by the rule probabilities), and for each nonterminal rule the three cases above. This gives

$$\begin{aligned} c(w|X) &= P(X \to w) \\ &\quad + \sum_{X \to YZ} P(X \to YZ) \\ &\qquad \Big(c(w|Y) + c(w|Z) \\ &\qquad\quad + \sum_{j=1}^{n-1} P(Y \stackrel{*}{\Rightarrow}_R w_1 \dots w_j) \\ &\qquad\qquad P(Z \stackrel{*}{\Rightarrow}_L w_{j+1} \dots w_n) \Big) \end{aligned} \tag{7.3}$$

In the important special case of bigrams, this summation simplifies quite a bit, since the terminal productions are ruled out and splitting into prefix and suffix allows but one possibility:

$$\begin{aligned} c(w_1 w_2|X) &= \sum_{X \to YZ} P(X \to YZ) \\ &\qquad \Big(c(w_1 w_2|Y) + c(w_1 w_2|Z) \\ &\qquad\quad + P(Y \stackrel{*}{\Rightarrow}_R w_1) P(Z \stackrel{*}{\Rightarrow}_L w_2) \Big) \end{aligned} \tag{7.4}$$

For unigrams equation (7.3) simplifies even more:

$$\begin{aligned} c(w_1|X) &= P(X \to w_1) \\ &\quad + \sum_{X \to YZ} P(X \to YZ) \Big(c(w_1|Y) + c(w_1|Z) \Big) \end{aligned} \tag{7.5}$$

We now have a recursive specification of the quantities $c(w|X)$ we need to compute. Alas, the recursion does not necessarily bottom out, since the $c(w|Y)$ and $c(w|Z)$ quantities on the right side of equation (7.3) may depend themselves on $c(w|X)$. Fortunately, the recurrence is linear, so for each string w, we can find the solution by solving the linear system formed by all equations of type (7.3). Notice there are exactly as many equations as variables, equal to the number of nonterminals in the grammar. The solution of these systems is further discussed below.

7.3.4 Computing prefix and suffix probabilities

The only substantial problem left at this point is the computation of the constants in equation (7.3). These are derived from the rule probabilities $P(X \to w)$ and $P(X \to YZ)$, as well as the prefix/suffix generation probabilities $P(Y \stackrel{*}{\Rightarrow}_R w_1 \dots w_j)$ and $P(Z \stackrel{*}{\Rightarrow}_L w_{j+1} \dots w_n)$.

The computation of prefix probabilities for SCFGs is generally useful for applications and has been solved with the LRI algorithm (Jelinek & Lafferty 1991). In Chapter 6 we have seen how this computation can be carried out efficiently for sparsely parameterized SCFGs using a probabilistic version of Earley's parser. Computing suffix probabilities is obviously a symmetrical task; for example, one could create a 'mirrored' SCFG (reversing the order of right-hand side symbols in all productions) and then run any prefix probability computation on that mirror grammar.

Note that in the case of bigrams, only a particularly simple form of prefix/suffix probabilities are required, namely, the 'left-corner' and 'right-corner' probabilities, $P(X \stackrel{*}{\Rightarrow}_L w_1)$ and $P(Y \stackrel{*}{\Rightarrow}_R w_2)$, which can each be obtained from a single matrix inversion (Jelinek & Lafferty 1991), corresponding to the left-corner matrix R_L used in the probabilistic Earley parser (as well as the corresponding right-corner matrix).

Finally, it is interesting to compare the relative ease with which one can solve the substring *expectation* problem to the seemingly similar problem of finding substring *probabilities*: the probability that X generates (one or more instances of) w. The latter problem is studied by Corazza *et al.* (1991), and shown to lead to a *non-linear* system of equations. The crucial difference here is that expectations are additive with respect to the cases in Figure 7.1, whereas the corresponding probabilities are not, since the three cases can occur in the same string.

7.3.5 N-grams containing string boundaries

A complete n-gram grammar includes strings delimited by a special marker denoting beginning and end of a string. In Section 2.2.2 we had introduced the symbol '\$' for this purpose.

To generate expectations for $n - 1$-grams adjoining the string boundaries, the original SCFG grammar is *augmented* by a new top-level production

$$S' \rightarrow \$S\$ \quad [1.0]$$

where S is the old start symbol, and S' becomes the start symbol of the augmented grammar. The algorithm is then simply applied to the augmented grammar to give the desired n-gram probabilities including the '\$' marker.

7.4 Efficiency and Complexity Issues

Summarizing from the previous section, we can compute any n-gram probability by solving two linear systems of equations of the form (7.3), one with w being the n-gram itself and one for the $(n-1)$-gram prefix $w_1 \ldots w_{n-1}$. The latter computation can be shared among all n-grams with the same prefix, so that essentially one system needs to be solved for each n-gram we are interested in. The good news here is that the work required is linear in the number of n-grams, and correspondingly limited if one needs probabilities for only a subset of the possible n-grams. For example, one could compute these probabilities on demand and cache the results.

Let us examine these systems of equations one more time. Each can be written in matrix notation in the form

$$(\mathbf{I} - \mathbf{A})\mathbf{c} = \mathbf{b} \tag{7.6}$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{A} = (a_{XU})$ is a coefficient matrix, $\mathbf{b} = (b_X)$ is the right-hand side vector, and $\mathbf{c}$ represents the vector of unknowns, $c(w|X)$. All of these are indexed by nonterminals X, U.

We get

$$\begin{aligned} a_{XU} &= \sum_{X \to YZ} P(X \to YZ)(\delta(Y,U) + \delta(Z,U)) & (7.7) \\ b_X &= P(X \to w) \\ &\quad + \sum_{X \to YZ} P(X \to YZ) \\ &\qquad\qquad \sum_{j=1}^{n-1} P(Y \stackrel{*}{\Rightarrow}_R w_1 \ldots w_j) \\ &\qquad\qquad\quad P(Z \stackrel{*}{\Rightarrow}_L w_{j+1} \ldots w_n) & (7.8) \end{aligned}$$

where $\delta(X, Y) = 1$ if $X = Y$, and 0 otherwise. The expression $\mathbf{I} - \mathbf{A}$ arises from bringing the variables $c(w|Y)$ and $c(w|Z)$ to the other side in equation (7.3) in order to collect the coefficients.

We can see that all dependencies on the particular bigram, w, are in the right-hand side vector $\mathbf{b}$, while the coefficient matrix $\mathbf{I} - \mathbf{A}$ depends only on the grammar. This, together with the standard method of *LU decomposition* (see, e.g., Press *et al.* (1988)) enables us to solve for each bigram in time $O(N^2)$, rather than the standard $O(N^3)$ for a full system (N being the number of nonterminals/variables). The LU decomposition itself is cubic, but is incurred only once. The full computation is therefore dominated by the quadratic effort of solving the system for each n-gram. Furthermore, the quadratic cost is a worst-case figure that would be incurred only if the grammar contained every possible rule; empirically this computation is linear in the number of nonterminals, for grammars that are *sparse*, i.e., where each nonterminal makes reference only to a bounded number of other nonterminals (independent of the total grammar size).

7.5 Consistency of SCFGs

Blindly applying the n-gram algorithm (and many others) to a SCFG with arbitrary probabilities can lead to surprising results. Consider the following simple grammar

$$\begin{array}{lcll} S & \to & x & [p] \\ S & \to & SS & [q = 1 - p] \end{array} \tag{7.9}$$

What is the expected frequency of unigram x? Using the abbreviation $c = c(X|S)$ and equation 7.5, we see that

$$\begin{aligned} c &= P(S \to x) + P(S \to SS)(c + c) \\ &= p + 2qc \end{aligned}$$

This leads to

$$c = \frac{p}{1-2q} = \frac{p}{2p-1}. \tag{7.10}$$

Now, for $p = 0.5$ this becomes infinity, and for probabilities $p < 0.5$, the solution is negative! This is a rather striking manifestation of the failure of this grammar, for $p \leq 0.5$, to be *consistent* in the sense of Booth & Thompson (1973) (see Section 6.4.8). An inconsistent grammar is one in which the stochastic derivation process has non-zero probability of not terminating. The expected length of the generated strings should therefore be infinite in this case.

Booth and Thompson derive a criterion for checking the consistency of a SCFG: Find the first-moment matrix $\mathbf{E} = (e_{XY})$, where e_{XY} is the expected number of occurrences of nonterminal Y in a one-step expansion of nonterminal X, and make sure its powers $\mathbf{E}^k$ converge to 0 as $k \to \infty$. If so, the grammar is consistent, otherwise it is not.[5]

For the grammar in (7.9), $\mathbf{E}$ is the 1×1 matrix $(2q)$. Thus we can confirm our earlier observation by noting that $(2q)^k$ converges to 0 iff $q < 0.5$, or $p > 0.5$.

Notice that $\mathbf{E}$ is identical to the matrix $\mathbf{A}$ that occurs in the linear equations (7.6) for the n-gram computation. The actual coefficient matrix is $\mathbf{I} - \mathbf{A}$, and its inverse, if it exists, can be written as the geometric sum

$$(\mathbf{I} - \mathbf{A})^{-1} = \mathbf{I} + \mathbf{A} + \mathbf{A}^2 + \mathbf{A}^3 + \ldots$$

This series converges precisely if $\mathbf{A}^k$ converges to 0. We have thus shown that the existence of a solution for the n-gram problem is equivalent to the consistency of the grammar in question. Furthermore, the solution vector $\mathbf{c} = (\mathbf{I} - \mathbf{A})^{-1}\mathbf{b}$ will always consist of non-negative numbers: it is the sum and product of the non-negative values given by equations (7.7) and (7.8).

The matrix $\mathbf{I} - \mathbf{A}$ and its inverse turn out to have a special role for SCFG: it is, in a sense, a 'universal problem solver' for a whole series of global quantities associated with probabilistic grammars. A brief overview of these is given in the appendix to this chapter.

7.6 Experiments

The algorithm described here has been implemented, and is being used to generate bigrams for a speech recognizer that is part of the BeRP spoken-language system (Jurafsky *et al.* 1994a). The speech decoder and language model components of the BeRP system were used in an experiment to assess the benefit of using bigram probabilities obtained through SCFGs versus estimating them directly from the available training corpus. The system's domain are inquiries about restaurants in the city of Berkeley. Table 7.1 gives statistics for the training and test corpora used, as well as the language models involved in the experiment. Our experiments made use of a context-free grammar hand-written for the BeRP domain. Computing the bigram probabilities from this SCFG of 133 nonterminals involves solving 657 linear systems for unigram

[5] An alternative version of this criterion is to check the magnitude of the largest of $\mathbf{E}$'s eigenvalues (its spectral radius). If that value is > 1, the grammar is inconsistent; if < 1, it is consistent.

	Training corpus	Test corpus
No. of sentences	2621	364
No. of words	16974	2208
Bigram vocabulary	1064	
Bigram coverage	100%	77%
SCFG productions	1177	
SCFG vocabulary	655	
SCFG coverage	63%	51%

Table 7.1: BeRP corpora and language model statistics.
Coverage is measured by the percentage of sentences parsed with non-zero probability by a given language model.

expectations and 108959 linear systems for bigram expectations. The process takes about 9 hours on a SPARCstation 10 using a non-optimized Lisp implementation.[6]

The experiments and results described below overlap with those reported in Jurafsky *et al.* (1994b).

In experiment 1, the recognizer used bigrams that were estimated directly from the training corpus, without any smoothing, resulting in a word error rate of 33.7%.

In experiment 2, a different set of bigram probabilities was used, computed from the context-free grammar, whose probabilities had previously been estimated from the same training corpus, using standard EM techniques. This resulted in a word error rate of 32.9%. This may seem surprisingly good given the low coverage of the underlying CFGs, but notice that the conversion into bigrams is bound to result in a less constraining language model, effectively increasing coverage. For comparison purposes we also ran the same experiment with bigrams computed indirectly by Monte-Carlo sampling from the SCFG, using 200,000 samples. The result was slightly worse (33.3%), confirming that the precise computation has an inherent advantage, as it cannot omit words or constructions that the SCFG assigns very low probability.

Finally, in experiment 3, the bigrams generated from the SCFG were augmented by those from the raw training data, in a proportion of 200,000 : 2500. We have not attempted to optimize this mixture proportion, e.g., by deleted interpolation (Jelinek & Mercer 1980).[7] With the bigram estimates thus obtained, the word error rate dropped to 29.6%, which represents a statistically significant improvement over experiments 1 and 2.

Table 7.2 summarizes these figures and also adds two more points of comparison: a pure SCFG language model and a mixture model that interpolates between bigram and SCFG. Notice that the latter case is different from experiment 3, where the language model used is a standard bigram, albeit one that was obtained by 'mixing' counts obtained both from the data and from the SCFG. The system referred to here, on

[6] One inefficiency is that the actual number of nonterminals (and hence the rank of the coefficient matrix) is 445, as the grammar is converted to the Simple Normal Form introduced in Chapter 4.

[7] This proportion comes about because in the original system, predating the method described here, bigrams had to be estimated from the SCFG by random sampling. Generating 200,000 sentence samples was found to give good converging estimates for the bigrams. The bigrams from the raw training sentences were then simply added to the randomly generated ones. We later verified that the bigrams estimated from the SCFG were indeed identical to the ones computed directly using the method described here.

	Word error (%)
Bigram estimated from raw data	33.7
Bigram computed from SCFG	32.9
by Monte-Carlo sampling	33.3
Bigram from SCFG plus data	29.6
SCFG	29.6
Mixture Bigram-SCFG	28.8

Table 7.2: Speech recognition accuracy using various language models.

the other hand, is a standard weighted mixture of two distinct submodels, as described in Section 2.3.1.

The experiments therefore support the argument made earlier that more sophisticated language models, even if far from perfect, can improve n-gram estimates obtained directly from sample data. We also see that the bulk of the improvement does not come from using a SCFG alone, but from smoothing the bigram statistics through the constraints imposed by the SCFG (possibly combined with a mixture of the two language models).

7.7 Summary

We have described an algorithm to compute in closed form the distribution of n-grams for a probabilistic language given by a stochastic context-free grammar. The algorithm is based on computing substring expectations, which can be expressed as systems of linear equations derived from the grammar. Listed below are the steps of the complete n-gram-from-SCFG computation. For concreteness we give the version specific to bigrams ($n = 2$).

1. Compute the prefix (left-corner) and suffix (right-corner) probabilities for each (nonterminal,word) pair.
2. Compute the coefficient matrix and right-hand sides for the systems of linear equations, as per equations (7.4) and (7.5).
3. LU decompose the coefficient matrix.
4. Compute the unigram expectations for each word in the grammar, by solving the LU system for the unigram right-hand sides computed in step 2.
5. Compute the bigram expectations for each word pair by solving the LU system for the bigram right-hand sides computed in step 2.
6. Compute each bigram probability $P(w_2|w_1)$, by dividing the bigram expectation $c(w_1 w_2|S)$ by the unigram expectation $c(w_1|S)$.

The algorithm has been found practical in the context of a medium-scale speech understanding system, were it gave improved estimates for a bigram language model, based on a hand-written SCFG and very small amounts of available training data.

Deriving n-gram probabilities from more sophisticated language models appears to be a generally useful technique which can both improve upon direct estimation of n-grams, and allows available higher-level linguistic knowledge to be effectively integrated into speech decoding or other tasks that place strong constraints on usable language models.

7.8 Appendix: Related Problems

We have seen how n-gram expectations for SCFGs can be obtained by solving linear systems based on the matrix $\mathbf{I} - \mathbf{A}$, where $\mathbf{I}$ is the identity matrix and $\mathbf{A}$ is the first-moment (expectation) matrix of nonterminal occurrences for a single nonterminal expansion. As it turns out, a number of apparently unrelated problems arising in connection with SCFGs and other probabilistic grammars have solutions based on this same matrix. These are briefly surveyed below, without detailed proofs.

7.8.1 Expected string length

To compute the expected number of terminals in a string, the system $\mathbf{I}-\mathbf{A}$ is solved for the right-hand side vector containing the average number of terminals generated in a single production, for each nonterminal.

For example, if

$$
\begin{aligned}
X &\rightarrow abZ \\
&\rightarrow c
\end{aligned}
$$

are all productions with LHS X, the entry indexed by X in the right-hand side vector would be $P(X \rightarrow abZ) \times 2 + P(X \rightarrow c) \times 1$.

The solution vector contains the expected lengths for the sublanguages generated by each of the nonterminals. Thus, the expected sentence string length is the S-entry in the solution vector.

The problems and its solution are easily generalized to obtain the expected number of terminals of a particular type occuring in a string (Booth & Thompson 1973).

7.8.2 Derivation entropy

The *derivation entropy* is the average number of bits required to specify a derivation from a SCFG. Is is computed from a right-hand side vector that contains the average negative log probabilities for the productions of each LHS nonterminal. For example, based on

$$
\begin{aligned}
X &\rightarrow \lambda \\
&\rightarrow \mu
\end{aligned}
$$

the right-hand side entry for X is $-P(X \to \lambda) \log P(X \to \lambda) - P(X \to \mu) \log P(X \to \mu)$.

The derivation entropy of the grammar is the solution vector entry indexed by the start symbol.

7.8.3 Expected number of nonterminal occurrences

Consider the following problem: starting from a nonterminal X, how many nonterminals of type Y are generated? Let $\mathbf{N} = (n_{XY})$ be the matrix formed by these expectations (assuming they exist). It is easy to verify that $\mathbf{N}$ satisfies the familiar recurrence $\mathbf{N} = \mathbf{I} + \mathbf{A}\mathbf{N}$, from which we get $\mathbf{N} = (\mathbf{I} - \mathbf{A})^{-1}$.

Another way to look at this outcome is that the expected number of nonterminals of type Y, represented by the column vector $\mathbf{n}^Y$ in $\mathbf{N}$, is a problem of the same type as those described above:

$$(\mathbf{I} - \mathbf{A})\mathbf{n}^Y = \mathbf{i}^Y$$

where $\mathbf{i}^Y$ is the Y-column of the identity matrix.

In fact, the nonterminal expectations are the most general of all the problems of this type, in the following sense. All the other quantities considered here, and conversely all those with a linear system solution defined by a right-hand side vector $\mathbf{b}$ and the coefficient matrix $\mathbf{I} - \mathbf{A}$, can be represented as *weighted sums* over the nonterminal expectations $\mathbf{N}$, where the weighting is provided by the vector $\mathbf{b}$.

For example, the expected length x_X of the strings generated by nonterminal X can be seen (from first principles) to be the weighted sum

$$x_X = \sum_Y n_{XY} b_Y$$

where n_{XY} is average number of Y's generated by X, and b_Y is the expected number of terminals produced by Y (in a single derivation step). Written in matrix form this becomes

$$\mathbf{x} = \mathbf{N}\mathbf{b}$$

and using $\mathbf{N} = (\mathbf{I} - \mathbf{A})^{-1}$ we get the system used above (Section 7.8.1) for solving $\mathbf{x}$,

$$(\mathbf{I} - \mathbf{A})\mathbf{x} = \mathbf{b} \quad .$$

Thus, derivation entropy, n-gram expectations, etc. all turn out to be linear weighted sums of the nonterminal expectations $\mathbf{N}$. Needless to say, these quantities are well-defined if and only if the underlying SCFG is consistent.

7.8.4 Other grammar types

The significance of $\mathbf{I} - \mathbf{A}$ extends to all types of probabilistic grammars based on Markov or branching processes over states. For example, in HMMs all of the above computations apply when 'nonterminal' is replaced by 'state' where appropriate. The first-moment matrix $\mathbf{A}$ for HMMs is just the transition matrix, and $\mathbf{N} = (\mathbf{I} - \mathbf{A})^{-1}$ expresses the expected number of visits to a state.

Chapter 8

Future directions

In this final chapter we suggest a number of potential continuations of the work described in this dissertation. Probabilistic language modeling has certainly proven to be a theoretically and practically useful approach, including the model merging paradigm for structural learning studied here. It has given rise to concrete applications (in the case of HMMs) and gives new perspectives on some linguistic issues, as we have tried to show mainly in our discussion of the SCFG and PAG modeling frameworks.

However, and not surprisingly, the work to date raises more questions than it provides definitive answers. Apart from model-specific problems discussed in the preceding chapters, a number of more fundamental issues can be identified. Invariably, these lead to worthwhile avenues for future work.

8.1 Formal characterization of learning dynamics

It would be extremely useful to be able to predict the *sample complexity* of a given target grammar and merging-based learning algorithm, i.e., the number of samples required to reliably learn the target grammar. As a first step, this would include a prediction of how many samples are needed to make the target grammar (or one equivalent to it) the one with globally optimal posterior probability. A further, more difficult problem is to predict when a limited search procedure of specified type would be able to actually find that global optimum.

Within this general problem of sample complexity it is also interesting to draw a distinction between sample structure and sample distribution. In other words, how sensitive is a learner to a sample that is representative regarding the possible strings of the language, but exhibits different frequency statistics?[1]

[1] This question was pointed out by Robert Wilensky.

8.2 Noisy samples

A related problem concerns the potential presence of *noise* in the sample, i.e., samples randomly altered by an independent process, such as random replacement of symbols. Plain model merging algorithms will simply try to model the language resulting from the composition of the undistorted language and the noise process. If the probabilities of distortions by noise are small enough one could expect to recover the original language model by *pruning* low probability parts of the induced model structures.

Pruning techniques have indeed been applied successfully in conjunction with the HMM merging algorithm as applied to multiple-pronunciation models (Section 3.6.2) in cases were the training corpus was known to contain outlier (mislabeled) samples, as reported by Wooters (1993). Again, a formal characterization of the conditions under which this method is successful would be obviously useful.

8.3 More informative search heuristics and biases

The work so far clearly shows that, as expected, increased model expressiveness requires a more varied repertoire of search operators, which in turn necessitates increasingly non-local search strategies. Carefully chosen heuristics and macro operators can counter the increase in search complexity in some cases.

This raises the question of what principled sources of *search bias* might be used. A main theme of the present studies was that a considerable amount of linguistic structure can be found by model merging, based solely on non-informative priors and 'dumb' search strategies. However, linguistic theories with strong *a priori* bases should help in cases where the uninformed approaches turn out to be insufficient. A prerequisite for this approach is that the predictions of the linguistic theory can be cast into effective search heuristics.

8.4 Induction by model specialization

Pure model merging is based on operators that leave the weak generative capacity of the models unchanged or produce a strictly more inclusive model. We already mentioned an alternative, inverse approach in Chapter 3: *model splitting* or *specialization*. Given the local nature of the search process, we expect improvements from adding operators that can effectively undo previous merging steps.

Also, a learning algorithm that goes from general-to-specific incorporates a different global bias in the face of local search: it will tend to err on the side of the too general, rather than the too specific model. Having both types of biases available allows choosing or mixing them according to the needs of particular applications. For example, if it is important that the resulting model have high coverage of new data, one might prefer over-general models.

The approaches in the literature that make use of state splitting are so far based entirely on likelihood criteria. However, the Bayesian evaluation methods used in this thesis are clearly separable from the merging and search components of the overall approach, and should apply to other systems as well. A first practical step towards a unified induction framework would therefore study how specializing algorithms can be combined with posterior probability maximization to advantage.

8.5 New applications

We already remarked that natural language is a less than ideal application domain for exclusively or mainly syntax-oriented probabilistic characterizations, due to the wide range of extra-syntactic constraints it is subject to. Formal language models are starting to be used more widely, in such areas as computational biology, graphical modeling ('picture grammars'), and document structure analysis. Probabilistic learning approaches, especially structural learning, still await study in these areas.

Still, natural language remains an important topic. The difficulties cited earlier seem to indicate that no single learning algorithm or paradigm can hope to be a practical way of inducing suitable models from large corpora. A natural strategy therefore is to make more selective, and combined use of partial solutions, as indicated at the end of Chapter 4. A major problem in this regard, apart from the obvious one of identifying the right partial solutions worth combining, will be to find the unifying principles that allow different approaches to 'talk' to one another. Here it seems that probability theory (and the derived information theoretic concepts, such as description length) can again play a central role.

8.6 New types of probabilistic models

Ultimately, probabilistic approaches to language need to identify alternatives to the traditional formalism used to date. We have seen that each such formalism defines itself by the underlying notion of derivation structure and the set of conditional independence assumptions made in defining the probabilities of derivations. In choosing these, there is always a design trade-off between capturing the relevant probabilistic contingencies found in a domain, and the computational expense of the associated algorithms for parsing, estimation, etc. It seems that properties of learnability (by model merging or otherwise), both complexity and robustness, should be added to these criteria. Model merging is a natural theoretical framework in this context, since it applies widely across model classes, and can thus serve as a basis for comparison in questions of learnability.

Bibliography

AHO, ALFRED V., RAVI SETHI, & JEFFREY D. ULLMAN. 1986. *Compilers: Principles, Techniques and Tools*. Reading, Mass.: Addison-Wesley.

——, & JEFFREY D. ULLMAN. 1972. *The Theory of Parsing, Translation, and Compiling. Volume1: Parsing*. Englewood Cliffs, N.J.: Prentice-Hall.

ANGLUIN, D., & C. H. SMITH. 1983. Inductive inference: Theory and methods. *ACM Computing Surveys* 15.237–269.

BAHL, LALIT R., FREDERICK JELINEK, & ROBERT L. MERCER. 1983. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 5.179–190.

BAKER, JAMES K. 1979. Trainable grammars for speech recognition. In *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, ed. by Jared J. Wolf & Dennis H. Klatt, 547–550, MIT, Cambridge, Mass.

BALDI, PIERRE, YVES CHAUVIN, TIM HUNKAPILLER, & MARCELLA A. MCCLURE. 1993. Hidden Markov models in molecular biology: New algorithms and applications. In *Advances in Neural Information Processing Systems 5*, ed. by Stephen José Hanson, Jack D. Cowan, & C. Lee Giles, 747–754. San Mateo, CA: Morgan Kaufmann.

BAUM, LEONARD E., TED PETRIE, GEORGE SOULES, & NORMAN WEISS. 1970. A maximization technique occuring in the statistical analysis of probabilistic functions in Markov chains. *The Annals of Mathematical Statistics* 41.164–171.

BELL, TIMOTHY C., JOHN G. CLEARY, & IAN H. WITTEN. 1990. *Text Compression*. Englewood Cliffs, N.J.: Prentice Hall.

BOOTH, TAYLOR L., & RICHARD A. THOMPSON. 1973. Applying probability measures to abstract languages. *IEEE Transactions on Computers* C-22.442–450.

BOURLARD, HERVÉ, & NELSON MORGAN. 1993. *Connectionist Speech Recognition. A Hybrid Approach*. Boston, Mass.: Kluwer Academic Publishers.

BRILL, ERIC. 1993. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings ARPA Workshop on Human Language Technology*, Plainsboro, N.J.

BRISCOE, TED, & JOHN CARROLL. 1993. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics* 19.25–59.

BROWN, PETER F., VINCENT J. DELLA PIETRA, PETER V. DESOUZA, JENIFER C. LAI, & ROBERT L. MERCER. 1992. Class-based n-gram models of natural language. *Computational Linguistics* 18.467–479.

BUNTINE, W. L. 1991. Theory refinement of Bayesian networks. In *Seventh Conference on Uncertainty in Artificial Intelligence*, Anaheim, CA.

BUNTINE, WRAY. 1992. Learning classification trees. In *Artificial Intelligence Frontiers in Statistics: AI and Statistics III*, ed. by D. J. Hand. Chapman & Hall.

CHEESEMAN, PETER, JAMES KELLY, MATTHEW SELF, JOHN STUTZ, WILL TAYLOR, & DON FREEMAN. 1988. AutoClass: A Bayesian classification system. In *Proceedings of the 5th International Conference on Machine Learning*, 54–64, University of Michigan, Ann Arbor, Mich.

CHEN, FRANCINE R. 1990. Identification of contextual factors for pronunciation networks. In *Proceedings IEEE Conference on Acoustics, Speech and Signal Processing*, volume 2, 753–756, Albuquerque, NM.

CHURCH, KENNETH W., & WILLIAM A. GALE. 1991. A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language* 5.19–54.

CLEEREMANS, AXEL, 1991. *Mechanisms of Implicit Learning. A Parallel Distributed Processing Model of Sequence Acquisition*. Pittsburgh, Pa.: Department of Psychology, Carnegie Mellon University dissertation.

COOK, CRAIG M., AZRIEL ROSENFELD, & ALAN R. ARONSON. 1976. Grammatical inference by hill climbing. *Information Sciences* 10.59–80.

COOPER, GREGORY F., & EDWARD HERSKOVITS. 1992. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning* 9.309–347.

CORAZZA, ANNA, RENATO DE MORI, ROBERTO GRETTER, & GIORGIO SATTA. 1991. Computation of probabilities for an island-driven parser. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.936–950.

COVER, THOMAS M., & JOY A. THOMAS. 1991. *Elements of Information Theory*. New York: John Wiley and Sons, Inc.

COX, R. T. 1946. Probability, frequency and reasonable expectation. *American Journal of Physics* 14.

DAGAN, IDO, FERNANDO PEREIRA, & LILLIAN LEE. 1994. Similarity-based estimation of word coocurrence probabilities. In *Proceedings of the 31th Annual Meeting of the Association for Computational Linguistics*, New Mexico State University, Las Cruces, NM.

DE SAUSSURE, FERDINAND. 1916. *Cours de linguistique generale*. Paris: Payot.

DEMPSTER, A. P., N. M. LAIRD, & D. B. RUBIN. 1977. Maximum likelihood from incomplete data via the *EM* algorithm. *Journal of the Royal Statistical Society, Series B* 34.1–38.

EARLEY, JAY. 1970. An efficient context-free parsing algorithm. *Communications of the ACM* 6.451–455.

EVANS, T. G. 1971. Grammatical inference techniques in pattern analysis. In *Software engineering*, ed. by J. Tou, 183–202. New York: Academic Press.

FASS, LEONA F. 1983. Learning context-free languages from their structured sentences. *ACM SIGACT News* 15.24–35.

FELDMAN, J., G. LAKOFF, D. BAILEY, S. NARAYANAN, T. REGIER, & A. STOLCKE. 1994. L_0—the first four years. *AI Review* 8. Special issue on Integration of Natural Language and Vision Processing, to appear.

FELDMAN, JEROME A., GEORGE LAKOFF, ANDREAS STOLCKE, & SUSAN HOLLBACH WEBER. 1990. Miniature language acquisition: A touchstone for cognitive science. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society*, 686–693, MIT, Cambridge, Mass.

FILLMORE, CHARLES J. 1988. The mechanisms of "Construction Grammar". In *Proceedings of the Fourteenth Annual Meeting of the Berkeley Linguistics Society*, ed. by Shelley Axmaker, Annie Jaisser, & Helen Singmeister, 35–55, Berkeley, Ca.

FUJISAKI, T., F. JELINEK, J. COCKE, E. BLACK, & T. NISHINO. 1991. A probabilistic parsing method for sentence disambiguation. In *Current Issues in Parsing Technology*, ed. by Masaru Tomita, chapter 10, 139–152. Boston: Kluwer Academic Publishers.

GAROFOLO, J. S., 1988. *Getting Started with the DARPA TIMIT CD-ROM: an Acoustic Phonetic Continuous Speech Database*. National Institute of Standards and Technology (NIST), Gaithersburgh, Maryland.

GAUVAIN, JEAN-LUC, & CHIN-HIN LEE. 1991. Bayesian learning of Gaussian mixture densities for hidden Markov models. In *Proceedings DARPA Speech and Natural Language Processing Workshop*, 271–277. Pacific Grove, CA: Defence Advanced Research Projects Agency, Information Science and Technology Office.

GAZDAR, GERALD, E. KLEIN, G. K. PULLUM, & I. A. SAG. 1985. *Generalized Phrase Structure Grammar*. Cambridge, Mass.: Harvard University Press.

GEMAN, STUART, ELIE BIENENSTOCK, & RENÉ DOURSAT. 1992. Neural networks and the bias/variance dilemma. *Neural Computation* 4.1–58.

——, & DONALD GEMAN. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6.721–741.

GRAHAM, SUSAN L., MICHAEL A. HARRISON, & WALTER L. RUZZO. 1980. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems* 2.415–462.

GULL, S. F. 1988. Bayesian inductive inference and maximum entropy. In *Maximum Entropy and Bayesian Methods in Science and Engineering, Volume 1: Foundations*, ed. by G. J. Erickson & C. R. Smith, 53–74. Dordrecht: Kluwer.

HAUSSLER, DAVID, ANDERS KROGH, I. SAIRA MIAN, & KIMMEN SJÖLANDER. 1992. Protein modeling using hidden Markov models: Analysis of globins. Technical Report UCSC-CRL-92-23, Computer and Information Sciences, University of California, Santa Cruz, Ca. Revised Sept. 1992.

HINTON, GEOFFREY E., & TERRENCE J. SEJNOWSKI. 1986. Learning and relearning in boltzmann machines. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, ed. by David E. Rumelhart & James L. McClelland, volume 1: *Foundations*, 282–317. Cambridge, Mass.: Bradford Books (MIT Press).

HJELMSLEV, LOUIS. 1953. *Prolegomena to a theory of language*. Baltimore: Waverly Press. Translated by Francis J. Whitfield from the Danish original *Omkring sprogteoriens grundlaeggelse*, Kopenhagen, 1943.

HOPCROFT, JOHN E., & JEFFREY D. ULLMAN. 1979. *Introduction to Automata Theory, Languages, and Computation*. Reading, Mass.: Addison-Wesley.

HORNING, JAMES JAY. 1969. A study of grammatical inference. Technical Report CS 139, Computer Science Department, Stanford University, Stanford, Ca.

JELINEK, FREDERICK. 1985. Markov source modeling in text generation. In *The Impact of Processing Techniques on Communications*, ed. by J. K. Skwirzinski. Dordrecht: Nijhoff.

——, & JOHN D. LAFFERTY. 1991. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics* 17.315–323.

——, ——, & ROBERT L. MERCER. 1992. Basic methods of probabilistic context free grammars. In *Speech Recognition and Understanding. Recent Advances, Trends, and Applications*, ed. by Pietro Laface & Renato De Mori, volume F75 of *NATO Advanced Sciences Institutes Series*, 345–360. Berlin: Springer Verlag. Proceedings of the NATO Advanced Study Institute, Cetraro, Italy, July 1990.

——, & ROBERT L. MERCER. 1980. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings Workshop on Pattern Recognition in Practice*, 381–397, Amsterdam.

JONES, MARK A., & JASON M. EISNER. 1992a. A probabilistic parser and its applications. In *AAAI Workshop on Statistically-Based NLP Techniques*, 20–27, San Jose, CA.

——, & ——. 1992b. A probabilistic parser applied to software testing documents. In *Proceedings of the 8th National Conference on Artificial Intelligence*, 332–328, San Jose, CA. AAAI Press.

JURAFSKY, DANIEL, CHUCK WOOTERS, GARY TAJCHMAN, JONATHAN SEGAL, ANDREAS STOLCKE, ERIC FOSLER, & NELSON MORGAN. 1994a. The Berkeley Restaurant Project. In *Proceedings International Conference on Spoken Language Processing*, Yokohama.

——, CHUCK WOOTERS, GARY TAJCHMAN, JONATHAN SEGAL, ANDREAS STOLCKE, & NELSON MORGAN. 1994b. Integrating experimental models of syntax, phonology and accent/dialect in a speech recognizer. In *AAAI Workshop on the Integration of Natural Language and Speech Processing*, ed. by Paul McKevitt, Seattle, WA.

KAPLAN, RONALD M., & JOAN BRESNAN. 1982. Lexical functional grammar: A formal system for grammatical representation. In *The Mental Representation of Grammatical Relations*, ed. by Joan Bresnan, 173–281. Cambridge, Mass.: MIT Press.

KATZ, SLAVA M. 1987. 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.

KUPIEC, JULIAN. 1992a. Hidden Markov estimation for unrestricted stochastic context-free grammars. In *Proceedings IEEE Conference on Acoustics, Speech and Signal Processing*, volume 1, 177–180, San Francisco.

——. 1992b. Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language* 6.225–242.

LANGLEY, PAT, 1994. Simplicity and representation change in grammar induction. Unpublished mss.

LARI, K., & S. J. YOUNG. 1990. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language* 4.35–56.

——, & ——. 1991. Applications of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language* 5.237–257.

LEE, H. C., & K. S. FU. 1972. Stochastic linguistics for picture recognition. Technical Report TR-EE 72-17, School of Electical Engineering, Purdue University.

MAGERMAN, DAVID M., & MITCHELL P. MARCUS. 1991. Pearl: A probabilistic chart parser. In *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics*, Berlin, Germany.

——, & CARL WEIR. 1992. Efficiency, robustness and accuracy in Picky chart parsing. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 40–47, University of Delaware, Newark, Delaware.

MITCHELL, TOM M. 1980. The need for biases in learning generalizations. Technical Report CBM-TR-117, Computer Science Department, Rutgers University, New Brunswick, N.J.

—— 1982. Generalization as search. *Artificial Intelligence* 18.203–226.

MORGAN, JAMES L., RICHARD P. MEIER, & ELISSA L. NEWPORT. 1987. Structural packaging in the input to language learning: Contributions of prosodic and morphological marking of phrases to the acquisition of language. *Cognitive Psychology* 19.498–550.

NAKAGAWA, SEI-ICHI. 1987. Spoken sentence recognition by time-synchronous parsing algorithm of context-free grammar. In *Proceedings IEEE Conference on Acoustics, Speech and Signal Processing*, volume 2, 829–832, Dallas, Texas.

NEAL, RADFORD M. 1993. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto.

NEY, HERMANN. 1984. The use of a one-stage dynamic programming algorithm for connected word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 32.263–271.

——. 1992. Stochastic grammars and pattern recognition. In *Speech Recognition and Understanding. Recent Advances, Trends, and Applications*, ed. by Pietro Laface & Renato De Mori, volume F75 of *NATO Advanced Sciences Institutes Series*, 319–344. Berlin: Springer Verlag. Proceedings of the NATO Advanced Study Institute, Cetraro, Italy, July 1990.

OMOHUNDRO, STEPHEN M. 1992. Best-first model merging for dynamic learning and recognition. Technical Report TR-92-004, International Computer Science Institute, Berkeley, Ca.

PÄSELER, ANNEDORE. 1988. Modification of Earley's algorithm for speech recognition. In *Recent Advances in Speech Understanding and Dialog Systems*, ed. by H. Niemann, M. Lang, & G. Sagerer, volume F46 of *NATO Advanced Sciences Institutes Series*, 466–472. Berlin: Springer Verlag. Proceedings of the NATO Advanced Study Institute, Bad Windsheim, Germany, July 1987.

PEARL, JUDEA. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, Ca.: Morgan Kaufman.

PEREIRA, FERNANDO, & YVES SCHABES. 1992. Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 128–135, University of Delaware, Newark, Delaware.

PORAT, SARA, & JEROME A. FELDMAN. 1991. Learning automata from ordered examples. *Machine Learning* 7.109–138.

PRESS, WILLIAM H., BRIAN P. FLANNERY, SAUL A. TEUKOLSKY, & WILLIAM T. VETTERLING. 1988. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press.

QUINLAN, J. ROSS, & RONALD L. RIVEST. 1989. Inferring decision trees using the minimum description length principle. *Information and Computation* 80.227–248.

RABINER, L. R., & B. H. JUANG. 1986. An introduction to hidden Markov models. *IEEE ASSP Magazine* 3.4–16.

REBER, A. S. 1969. Implicit learning of artifical grammars. *Journal of Verbal Learning and Verbal Behavior* 6.855–863.

REDNER, RICHARD A., & HOMER F. WALKER. 1984. Mixture densities, maximum likelihood and the EM algorithm. *SIAM Review* 26.195–239.

REGIER, TERRY, 1992. *The acquisition of lexical semantics for spatial terms: A connectionist model of perceptual categorization*. Berkeley, Ca.: Computer Science Division, University of California dissertation.

RESNIK, PHILIP. 1992. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings of the 14th International Conference on Computational Linguistics*, 418–424, Nantes, France.

RICHARDS, I. A., & C. GIBSON. 1945. *English Through Pictures*. New York: Washington Square Press.

RILEY, MICHAEL D. 1991. A statistical model for generating pronunciation networks. In *Proceedings IEEE Conference on Acoustics, Speech and Signal Processing*, volume 2, 737–740, Toronto.

RISSANEN, JORMA. 1983. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics* 11.416–431.

RON, DANA, YORAM SINGER, & NAFTALI TISHBY. 1994. The power of amnesia. In *Advances in Neural Information Processing Systems 6*, ed. by Jack Cowan, Gerald Tesauro, & Joshua Alspector. San Mateo, CA: Morgan Kaufmann.

SAKAKIBARA, YASUBUMI. 1990. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science* 76.223–242.

——, MICHAEL BROWN, RICHARD HUGHEY, I. SAIRA MIAN, KIMMEN SJÖLANDER, & REBECCA C. UNDERWOOD DAVID HAUSSLER. 1994. The application of stochastic context-free grammars to folding, aligning and modeling homologous RNA sequences. Technical Report UCSC-CRL-94-14, Computer and Information Sciences, University of California, Santa Cruz, Ca.

SCHABES, YVES, 1991. An inside-outside algorithm for estimating the parameters of a hidden stochastic context-free grammar based on Earley's algorithm. Unpublished mss. Presented at the Second Workshop on Mathematics of Language, Tarritown, N.Y., May 1991.

——. 1992. Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the 14th International Conference on Computational Linguistics*, 426–433, Nantes, France.

SCHWARTZ, RICHARD, & YEN-LU CHOW. 1990. The N-best algorithm: An efficient and exact procedure for finding the n most likely sentence hypotheses. In *Proceedings IEEE Conference on Acoustics, Speech and Signal Processing*, volume 1, 81–84, Albuquerque, NM.

SHIEBER, STUART M. 1986. *An Introduction to Unification-Based Approaches to Grammar*. Number 4 in CSLI Lecture Note Series. Stanford, Ca.: Center for the Study of Language and Information.

STOLCKE, ANDREAS. 1993. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. Technical Report TR-93-065, International Computer Science Institute, Berkeley, CA. To appear in *Computational Linguistics*.

——. 1994. How to BOOGIE: a manual for Bayesian, object-oriented, grammar induction and estimation. Internal memo, International Computer Science Institute.

——, & STEPHEN OMOHUNDRO. 1993. Hidden Markov model induction by Bayesian model merging. In *Advances in Neural Information Processing Systems 5*, ed. by Stephen José Hanson, Jack D. Cowan, & C. Lee Giles, 11–18. San Mateo, CA: Morgan Kaufmann.

——, & STEPHEN OMOHUNDRO. 1994a. Best-first model merging for hidden Markov model induction. Technical Report TR-94-003, International Computer Science Institute, Berkeley, CA.

——, & STEPHEN OMOHUNDRO. 1994b. Inducing probabilistic grammars by Bayesian model merging. In *Grammatical Inference and Applications. Second International Colloquium on Grammatical Inference*, Alicante, Spain. Springer Verlag. To appear.

——, & JONATHAN SEGAL. 1994. Precise n-gram probabilities from stochastic context-free grammars. In *Proceedings of the 31th Annual Meeting of the Association for Computational Linguistics*, 74–79, New Mexico State University, Las Cruces, NM.

THOMASON, MICHAEL G., & ERIK GRANUM. 1986. Dynamic programming inference of Markov networks from finite set of sample strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8.491–501.

TOMITA, MASARU. 1982. Dynamic construction of finite automata from examples using hill-climbing. In *Proceedings of the 4th Annual Conference of the Cognitive Science Society*, 105–108, Ann Arbor, Mich.

——. 1986. *Efficient Parsing for Natural Language*. Boston: Kluwer Academic Publishers.

UNDERWOOD, REBECCA CHRISTINE. 1994. Stochastic context-free grammars for modeling three spliceosomal small nuclear ribonucleic acids. Technical Report CRL-94-23, Computer and Information Sciences, University of California, Santa Cruz, Ca.

VITERBI, A. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory* 13.260–269.

WALLACE, C. S., & P. R. FREEMAN. 1987. Estimation and inference by compact coding. *Journal of the Royal Statistical Society, Series B* 49.240–265.

WOLFF, J. G. 1978. Grammar discovery as data compression. In *Proceedings of the AISB/GI Conference on Artificial Intelligence*, 375–379, Hamburg.

WOOTERS, CHARLES C., 1993. *Lexical Modeling in a Speaker Independent Speech Understanding System*. Berkeley, CA: University of California dissertation.

WOOTERS, CHUCK, & ANDREAS STOLCKE. 1994. Multiple-pronunciation lexical modeling in a speaker-independent speech understanding system. In *Proceedings International Conference on Spoken Language Processing*, Yokohama.

WRIGHT, J. H. 1990. LR parsing of probabilistic grammars with input uncertainty for speech recognition. *Computer Speech and Language* 4.297–323.

ZUE, VICTOR, JAMES GLASS, DAVID GOODINE, HONG LEUNG, MICHAEL PHILLIPS, JOSEPH POLIFRONI, & STEPHANIE SENEFF. 1991. Integration of speech recognition and natural language processing in the MIT Voyager system. In *Proceedings IEEE Conference on Acoustics, Speech and Signal Processing*, volume 1, 713–716, Toronto.