Stochastic Attribute-Value Grammars

Steven P. Abney*
AT&T Laboratories

Probabilistic analogues of regular and context-free grammars are well known in computational linguistics, and currently the subject of intensive research. To date, however, no satisfactory probabilistic analogue of attribute-value grammars has been proposed: previous attempts have failed to define an adequate parameter-estimation algorithm.

In the present paper, I define stochastic attribute-value grammars and give an algorithm for computing the maximum-likelihood estimate of their parameters. The estimation algorithm is adapted from Della Pietra, Della Pietra, and Lafferty (1995). To estimate model parameters, it is necessary to compute the expectations of certain functions under random fields. In the application discussed by Della Pietra, Della Pietra, and Lafferty (representing English orthographic constraints), Gibbs sampling can be used to estimate the needed expectations. The fact that attribute-value grammars generate constrained languages makes Gibbs sampling inapplicable, but I show that sampling can be done using the more general Metropolis-Hastings algorithm.

1. Introduction

Stochastic versions of regular grammars and context-free grammars have received a great deal of attention in computational linguistics for the last several years, and basic techniques of stochastic parsing and parameter estimation have been known for decades. However, regular and context-free grammars are widely deemed linguistically inadequate; standard grammars in computational linguistics are attribute-value (AV) grammars of some variety. Before the advent of statistical methods, regular and context-free grammars were considered too inexpressive for serious consideration, and even now the reliance on stochastic versions of the less-expressive grammars is often seen as an expedient necessitated by the lack of an adequate stochastic version of attribute-value grammars.

Proposals have been made for extending stochastic models developed for the regular and context-free cases to grammars with constraints.[1] Brew (1995) sketches a probabilistic version of Head-Driven Phrase Structure Grammar (HPSG). He proposes a stochastic process for generating attribute-value structures, that is, directed acyclic graphs (dags). A dag is generated starting from a single node labeled with the (unique) most general type. Each type S has a set of **maximal subtypes** $T_1, \dots, T_m$. To expand a node labeled S, one chooses a maximal subtype T stochastically. One then considers equating the current node with other nodes of type T, making a stochastic yes/no de-

* AT&T Laboratories, Rm. A249, 180 Park Avenue, Florham Park, NJ 07932

[1] I confine my discussion here to Brew and Eisele because they aim to describe parametric models of probability distributions over the languages of constraint-based grammars, and to estimate the parameters of those models. Other authors have assigned weights or preferences to constraint-based grammars but not discussed parameter estimation. One approach of the latter sort that I find of particular interest is that of Stefan Riezler (Riezler 1996), who describes a weighted logic for constraint-based grammars that characterizes the languages of the grammars as fuzzy sets. This interpretation avoids the need for normalization that Brew and Eisele face, though parameter estimation still remains to be addressed.

cision for each. Equating two nodes creates a re-entrancy. If the current node is equated with no other node, one proceeds to expand it. Each maximal type **introduces** types $U_1, \ldots, U_n$, corresponding to values of attributes; one creates a child node for each introduced type, and then expands each child in turn. A limitation of this approach is that it permits one to specify only the average rate of re-entrancies; it does not permit one to specify more complex context dependencies.

Eisele (1994) takes a logic-programming approach to constraint grammars. He assigns probabilities to proof trees by attaching parameters to logic program clauses. He presents the following logic program as an example:

1. $\texttt{p(X,Y,Z)} \leftarrow_1 \texttt{q(X,Y), r(Y,Z).}$
2. $\texttt{q(a,b)} \leftarrow_{0.4} \texttt{.}$
3. $\texttt{q(X,c)} \leftarrow_{0.6} \texttt{.}$
4. $\texttt{r(b,d)} \leftarrow_{0.5} \texttt{.}$
5. $\texttt{r(X,e)} \leftarrow_{0.5} \texttt{.}$

The probability of a proof tree is defined to be proportional to the product of the probabilities of clauses used in the proof. Normalization is necessary because some derivations lead to invalid proof trees. For example, the derivation

$$\texttt{p(X,Y,Z)} \xleftarrow{by\ 1} \texttt{q(X,Y) r(Y,Z)} \xleftarrow{by\ 3} \texttt{r(c,Z) : Y=c} \xleftarrow{by\ 4} \texttt{: Y=c b=c Z=d}$$

is invalid because of the illegal assignment $b = c$.

Both Brew and Eisele associate weights with analogues of rewrite rules. In Brew's case, we can view type expansion as a stochastic choice from a finite set of rules of form $X \rightarrow \xi_i$, where X is the type to expand and each ξ_i is a sequence of introduced child types. A re-entrancy decision is a stochastic choice between two rules, $X \rightarrow$ *yes* and $X \rightarrow$ *no*, where X is the type of the node being considered for re-entrancy. In Eisele's case, expanding a goal term can be viewed as a stochastic choice among a finite set of rules $X \rightarrow \xi_i$, where X is the predicate of the goal term and each ξ_i is a program clause whose head has predicate X. The parameters of the models are essentially weights on such rules, representing the probability of choosing ξ_i when making a choice of type X.

In these terms, Brew and Eisele propose estimating parameters as the empirical relative frequency of the corresponding rules. That is, the weight of the rule $X \rightarrow \xi_i$ is obtained by counting the number of times X rewrites as ξ_i in the training corpus, divided by the total number of times X is rewritten in the training corpus. For want of a standard term, let us call these estimates Empirical Relative Frequency (ERF) estimates. To deal with incomplete data, both Brew and Eisele appeal to the Expectation-Maximization (EM) algorithm, applied however to ERF rather than maximum-likelihood estimates.

Under certain independence conditions, ERF estimates *are* maximum-likelihood estimates. Unfortunately, these conditions are violated when there are context dependencies of the sort found in attribute-value grammars, as will be shown below. As a consequence, applying the ERF method to attribute-value grammars does *not* generally yield maximum-likelihood estimates. This is true whether one uses EM or not—a method that yields the "wrong" estimates on complete data does not improve when EM is used to extend the method to incomplete data.

Eisele identifies an important symptom that something is amiss with ERF estimates: the probability distribution over proof trees that one obtains does not agree with the frequency of proof trees in the training corpus. Eisele recognizes that this problem arises only where there are context dependencies.

Fortunately, solutions to the context-dependency problem have been described (and indeed are currently enjoying a surge of interest) in statistics, machine learning, and statistical pattern recognition, particularly image processing. The models of interest are known as random fields. Random fields can be seen as a generalization of Markov chains and stochastic branching processes. Markov chains are stochastic processes corresponding to regular grammars and random branching processes are stochastic processes corresponding to context-free grammars. The evolution of a Markov chain describes a line, in which each stochastic choice depends only on the state at the immediately preceding time-point. The evolution of a random branching process describes a tree in which a finite-state process may spawn multiple child processes at the next time-step, but the number of processes and their states depend only on the state of the unique parent process at the preceding time-step. In particular, stochastic choices are *independent* of other choices at the same time-step: each process evolves independently. If we permit re-entrancies, that is, if we permit processes to re-merge, we generally introduce context-sensitivity. In order to re-merge, processes must be "in synch," which is to say, they cannot evolve in complete independence of one another. Random fields are a particular class of multidimensional random processes, that is, processes corresponding to probability distributions over an arbitrary graph. The theory of random fields can be traced back to Gibbs (1902); indeed, the probability distributions involved are known as Gibbs distributions.

To my knowledge, the first application of random fields to natural language was Mark et al. (1992). The problem of interest was how to combine a stochastic context-free grammar with n-gram language models. In the resulting structures, the probability of choosing a particular word is constrained simultaneously by the syntactic tree in which it appears and the choices of words at the n preceding positions. The context-sensitive constraints introduced by the n-gram model are reflected in re-entrancies in the structure of statistical dependencies, as in Figure 1.

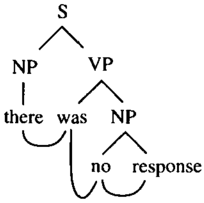


Figure 1
Statistical dependencies under the model of Mark et al. (1992).

In this diagram, the choice of label on a node z with parent x and preceding word y is dependent on the label of x and y, but conditionally independent of the label on any other node.

Della Pietra, Della Pietra, and Lafferty (1995, henceforth, DD&L) also apply random fields to natural language processing. The application they consider is the induction of English orthographic constraints—inducing a grammar of possible English words. DD&L describe an algorithm called Improved Iterative Scaling (IIS) for selecting informative features of words to construct a random field, and for setting the parameters of the field optimally for a given set of features, to model an empirical word distribution.

It is not immediately obvious how to use the IIS algorithm to equip attribute-value grammars with probabilities. In brief, the difficulty is that the IIS algorithm requires the computation of the expectations, under random fields, of certain functions; in general, computing these expectations involves summing over all configurations (all possible character sequences, in the orthography application), which is not possible when the configuration space is large. Instead, DD&L use Gibbs sampling to estimate the needed expectations.

Gibbs sampling is possible for the application that DD&L consider. A prerequisite for Gibbs sampling is that the configuration space be closed under relabeling of graph nodes. In the orthography application, the configuration space is the set of possible English words, represented as finite linear graphs labeled with ASCII characters. Every way of changing a label, that is, every substitution of one ASCII character for a different one, yields a possible English word.

By contrast, the set of graphs admitted by an attribute-value grammar G is highly constrained. If one changes an arbitrary node label in a dag admitted by G, one does *not* necessarily obtain a new dag admitted by G. Hence, Gibbs sampling is not applicable. However, I will show that a more general sampling method, the Metropolis-Hastings algorithm, can be used to compute the maximum-likelihood estimate of the parameters of AV grammars.

2. Stochastic Context-Free Grammars

Let us begin by examining stochastic context-free grammars (SCFGs) and asking why the natural extension of SCFG parameter estimation to attribute-value grammars fails. A point of terminology: I will use the term **grammar** to refer to an unweighted grammar, be it a context-free grammar or attribute-value grammar. A grammar equipped with weights (and other periphenalia as necessary) I will refer to as a **model**. Occasionally I will also use model to refer to the weights themselves, or the probability distribution they define.

Throughout we will use the following stochastic context-free grammar for illustrative purposes. Let us call the underlying grammar G_1 and the grammar equipped with weights as shown, M_1:

1. $\mathrm{S} \rightarrow \mathrm{A}\ \mathrm{A} \quad \beta_1 = 1/2$
2. $\mathrm{S} \rightarrow \mathrm{B} \quad \beta_2 = 1/2$
3. $\mathrm{A} \rightarrow \mathrm{a} \quad \beta_3 = 2/3$
4. $\mathrm{A} \rightarrow \mathrm{b} \quad \beta_4 = 1/3$
5. $\mathrm{B} \rightarrow \mathrm{a}\ \mathrm{a} \quad \beta_5 = 1/2$
6. $\mathrm{B} \rightarrow \mathrm{b}\ \mathrm{b} \quad \beta_6 = 1/2$

The probability of a given tree is computed as the product of probabilities of rules used in it. For example: Let x be the tree in Figure 2 and let q_1 be the probability distribution over trees defined by model M_1. Then:

$$q_1(x) = \beta_1 \cdot \beta_3 \cdot \beta_3 = \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{2}{3} = \frac{2}{9}$$

In parsing, we use the probability distribution $q_1(x)$ defined by model M_1 to disambiguate: the grammar assigns some set of trees $\{x_1, \ldots, x_n\}$ to a sentence σ, and we

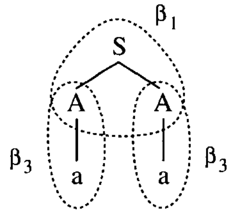


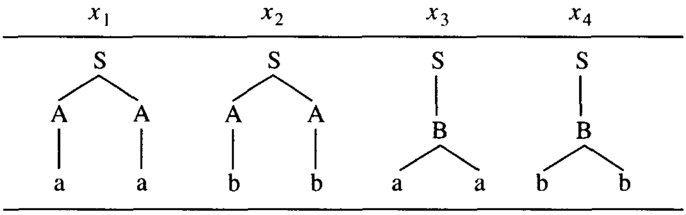
Figure 2
Computing the probability of a parse tree.

choose that tree x_i that has greatest probability $q_1(x_i)$. The issue of efficiently computing the most-probable parse for a given sentence has been thoroughly addressed in the literature. The standard parsing techniques can be readily adapted to the random-field models to be discussed below, so I simply refer the reader to the literature. Instead, I concentrate on parameter estimation, which, for attribute-value grammars, cannot be accomplished by standard techniques.

By parameter estimation we mean determining values for the weights β. In order for a stochastic grammar to be useful, we must be able to compute the correct weights, where by correct weights we mean the weights that best account for a training corpus. The degree to which a given set of weights accounts for a training corpus is measured by the similarity between the distribution $q(x)$ determined by the weights β and the distribution of trees x in the training corpus.

2.1 The Goodness of a Model

The distribution determined by the training corpus is known as the **empirical distribution**. For example, suppose we have a training corpus containing twelve trees of the four types from $L(G_1)$ shown in Figure 3, where $c(x)$ is the count of how often the tree (type) x appears in the corpus, and $\tilde{p}(\cdot)$ is the empirical distribution, defined as:



c =	4x	2x	3x	3x	= 12
p̃ =	4/12	2/12	3/12	3/12	

Figure 3
An empirical distribution. There are twelve parse trees of four distinct types.

$$\tilde{p}(x) = \frac{c(x)}{N} \qquad N = \sum_x c(x)$$

In comparing a distribution q to the empirical distribution $\tilde{p}$, we shall actually measure dissimilarity rather than similarity. Our measure for dissimilarity of distributions

Table 1
Computing the divergence of q_1 from $\tilde{p}$.

	q_1	$\tilde{p}$	$q_1/\tilde{p}$	$\tilde{p}\ln(\tilde{p}/q_1)$
x_1	2/9	1/3	0.67	0.14
x_2	1/18	1/6	0.33	0.18
x_3	1/4	1/4	1.00	0.00
x_4	1/4	1/4	1.00	0.00
				0.32

Table 2
Computing the divergence of q' from $\tilde{p}$.

	q'	$\tilde{p}$	$q'/\tilde{p}$	$\tilde{p}\ln(\tilde{p}/q')$
x_1	1/8	1/3	0.38	0.33
x_2	1/8	1/6	0.75	0.05
x_3	1/4	1/4	1.00	0.00
x_4	1/4	1/4	1.00	0.00
				0.38

is the Kullback-Leibler (KL) divergence, defined as:

$$D(\tilde{p}\|q) = \sum_x \tilde{p}(x) \ln \frac{\tilde{p}(x)}{q(x)}$$

The divergence between $\tilde{p}$ and q at point x is the log of the ratio of $\tilde{p}(x)$ to $q(x)$. The overall divergence between $\tilde{p}$ and q is the average divergence, where the averaging is over tree (tokens) in the corpus; i.e., point divergences $\ln(\tilde{p}(x)/q(x))$ are weighted by $\tilde{p}(x)$ and summed.

For example, let q_1 be, as before, the distribution determined by model M_1. Table 1 shows q_1, $\tilde{p}$, the ratio $q_1(x)/\tilde{p}(x)$, and the weighted point divergence $\tilde{p}(x)\ln(\tilde{p}(x)/q_1(x))$. The sum of the fourth column is the KL divergence $D(\tilde{p}\|q_1)$ between $\tilde{p}$ and q_1. The third column contains $q_1(x)/\tilde{p}(x)$ rather than $\tilde{p}(x)/q_1(x)$ so that one can see at a glance whether $q_1(x)$ is too large (> 1) or too small (< 1). The total divergence $D(\tilde{p}\|q_1) = 0.32$.

One set of weights is better than another if its divergence from the empirical distribution is less. For example, let us consider a different set of weights for grammar G_1. Let M' be G_1 with weights $(1/2, 1/2, 1/2, 1/2, 1/2, 1/2)$, and let q' be the probability distribution determined by M'. Then the computation of the KL divergence is as in Table 2. The fit for x_2 improves, but that is more than offset by a poorer fit for x_1. The distribution q_1 is a better distribution than q', in the sense that q_1 is more similar (less dissimilar) to the empirical distribution than q' is.

One reason for adopting minimal KL divergence as a measure of goodness is that minimizing KL divergence maximizes likelihood. The likelihood of distribution q is the probability of the training corpus according to q:

$$\begin{aligned} L(q) &= \prod_{x \textit{ in training}} q(x) \\ &= \prod_x q(x)^{c(x)} \end{aligned}$$

Since log is monotone increasing, maximizing likelihood is equivalent to maximizing log likelihood:

$$\begin{aligned}\ln L(q) &= \sum_x c(x) \ln q(x) \\ &= N \sum_x \tilde{p}(x) \ln q(x)\end{aligned}$$

The expression on the right-hand side is $-1/N$ times the cross entropy of q with respect to $\tilde{p}$, hence maximizing log likelihood is equivalent to minimizing cross entropy. Finally, $D(\tilde{p}\|q)$ is equal to the cross entropy of q less the entropy of $\tilde{p}$, and the entropy of $\tilde{p}$ is constant with respect to q; hence minimizing cross entropy (maximizing likelihood) is equivalent to minimizing divergence.

2.2 The ERF Method

For stochastic context-free grammars, it can be shown that the ERF method yields the best model for a given training corpus. First, let us introduce some terminology and notation. With each rule i in a stochastic context-free grammar is associated a weight β_i and a function $f_i(x)$ that returns the number of times rule i is used in the derivation of tree x. For example, consider the tree in Figure 2, repeated here in Figure 4 for convenience: Rule 1 is used once and rule 3 is used twice; accordingly $f_1(x) = 1$, $f_3(x) = 2$, and $f_i(x) = 0$ for $i \in \{2, 4, 5, 6\}$.

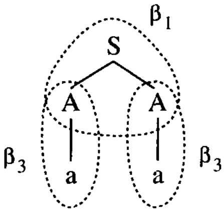


Figure 4
Rule applications in a parse tree.

We use the notation $p[f]$ to represent the expectation of f under probability distribution p; that is, $p[f] = \sum_x p(x)f(x)$. The ERF method instructs us to choose the weight β_i for rule i proportional to its empirical expectation $\tilde{p}[f_i]$. Algorithmically, we compute the expectation of each rule's frequency, and normalize among rules with the same left-hand side.

To illustrate, let us consider corpus (2.1) again. The expectation of each rule frequency f_i is a sum of terms $\tilde{p}(x)f_i(x)$. These terms are shown for each tree, in Table 3. For example, in tree x_1, rule 1 is used once and rule 3 is used twice. The empirical probability of x_1 is $1/3$, so x_1's contribution to $\tilde{p}[f_1]$ is $1/3 \cdot 1$, and its contribution to $\tilde{p}[f_3]$ is $1/3 \cdot 2$. The weight β_i is obtained from $\tilde{p}[f_i]$ by normalizing among rules with the same left-hand side. For example, the expected rule frequencies $\tilde{p}[f_1]$ and $\tilde{p}[f_2]$ of rules with left-hand side S already sum to 1, so they are adopted without change as β_1 and β_2. On the other hand, the expected rule frequencies $\tilde{p}[f_5]$ and $\tilde{p}[f_6]$ for rules with left-hand side B sum to $1/2$, not 1, so they are doubled to yield weights β_5 and β_6. It should be observed that the resulting weights are precisely the weights of model M_1.

It can be proven that the ERF weights are the best weights for a given context-free grammar, in the sense that they define the distribution that is most similar to the empirical distribution. That is, if β are the ERF weights (for a given grammar),

Table 3
Parameter estimation using the ERF method.

			S → A A	S → B	A → a	A → b	B → a a	B → b b
		$\tilde{p}$	$\tilde{p}f_1$	$\tilde{p}f_2$	$\tilde{p}f_3$	$\tilde{p}f_4$	$\tilde{p}f_5$	$\tilde{p}f_6$
x_1	$[_S\,[_A\,a]\,[_A\,a]]$	1/3	1/3		2/3			
x_2	$[_S\,[_A\,b]\,[_A\,b]]$	1/6	1/6			2/6		
x_3	$[_S\,[_B\,a\,a]]$	1/4		1/4			1/4	
x_4	$[_S\,[_B\,b\,b]]$	1/4		1/4				1/4
		$\tilde{p}[f] =$	1/2	1/2	2/3	1/3	1/4	1/4
		$\beta =$	1/2	1/2	2/3	1/3	1/2	1/2

defining distribution q, and β' defining q' is any set of weights such that $q \neq q'$, then $D(\tilde{p}\|q) < D(\tilde{p}\|q')$.

One might expect the best weights to yield $D(\tilde{p}\|q) = 0$, but such is not the case. We have just seen, for example, that the best weights for grammar G_1 yield distribution q_1, yet $D(\tilde{p}\|q_1) = 0.32 > 0$. A closer inspection of the divergence calculation in Table 1 reveals that q_1 is sometimes less than $\tilde{p}$, but never greater than $\tilde{p}$. Could we improve the fit by increasing q_1? For that matter, how can it be that q_1 is never greater than $\tilde{p}$? As probability distributions, q_1 and $\tilde{p}$ should have the same total mass, namely, one. Where is the missing mass for q_1?

The answer is of course that q_1 and $\tilde{p}$ are probability distributions over $L(G_1)$, but not all of $L(G_1)$ appears in the corpus. Two trees are missing, and they account for the missing mass. These two trees are given in Figure 5. Each of these trees has probability 0 according to $\tilde{p}$ (hence they can be ignored in the divergence calculation), but probability 1/9 according to q_1.

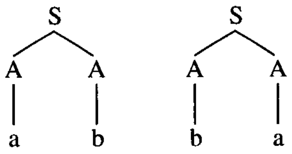


Figure 5
The trees from $L(G_1)$ that are missing in the training corpus.

Intuitively, the problem is this: The distribution q_1 assigns too little weight to trees x_1 and x_2, and too much weight to the "missing" trees; call them x_5 and x_6. Yet exactly the same rules are used in x_5 and x_6 as are used in x_1 and x_2. Hence there is no way to increase the weight for trees x_1 and x_2, improving their fit to $\tilde{p}$, without simultaneously increasing the weight for x_5 and x_6, making their fit to $\tilde{p}$ worse. The distribution q_1 is the best compromise possible.

To say it another way, our assumption that the corpus was generated by a context-free grammar means that any context dependencies in the corpus must be accidental, the result of sampling noise. There is indeed a dependency in the corpus in Figure 3: in the trees where there are two A's, the A's always rewrite the same way. If the corpus was generated by a stochastic context-free grammar, then this dependency is accidental.

This does not mean that the context-free assumption is wrong. If we generate twelve trees at random from q_1, it would not be too surprising if we got the corpus in Figure 3. More extremely, if we generate a random corpus of size 1 from q_1, it is quite impossible for the resulting empirical distribution to match the distribution q_1. But as the corpus size increases, the fit between $\tilde{p}$ and q_1 becomes ever better.

3. Attribute-Value Grammars

But what if the dependency in corpus (3) is not accidental? What if we wish to adopt a grammar that imposes the constraint that both A's rewrite the same way? We can impose such a constraint by means of an attribute-value grammar.

We may formalize an attribute-value grammar as a context-free grammar with attribute labels and path equations. An example is the following grammar; let us call it G_2:

1. $\mathrm{S} \rightarrow 1{:}\mathrm{A}\ 2{:}\mathrm{A}\ \ \langle 1\ 1\rangle = \langle 2\ 1\rangle$
2. $\mathrm{S} \rightarrow 1{:}\mathrm{B}$
3. $\mathrm{A} \rightarrow 1{:}\mathrm{a}$
4. $\mathrm{A} \rightarrow 1{:}\mathrm{b}$
5. $\mathrm{B} \rightarrow 1{:}\mathrm{a}$
6. $\mathrm{B} \rightarrow 1{:}\mathrm{b}$

Figure 6 illustrates how a dag is generated from G_2. We begin in (a) with a single node labeled with the start category of G_2, namely, S. A node x is expanded by choosing a rule that rewrites the category of x. In this case, we choose rule *1* to expand the root node. Rule 1 instructs us to create two children, both labeled A. The edge to the first child is labeled *1* and the edge to the second child is labeled *2*. The constraint $\langle 1\ 1\rangle = \langle 2\ 1\rangle$ indicates that the *1* child of the *1* child of x is identical to the *1* child of the *2* child of x. We create an unlabeled node to represent this grandchild of x and direct appropriately labeled edges from the children, yielding (b).

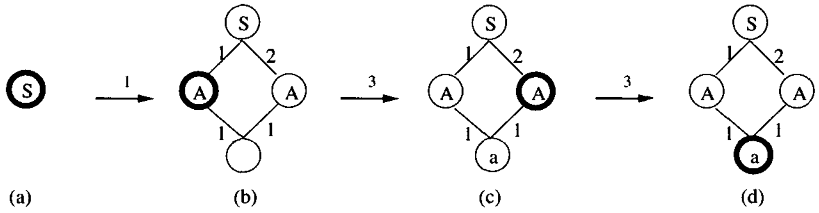


Figure 6
Generating a dag. The grammar used is G_2.

We proceed to expand the newly introduced nodes. We choose rule 3 to expand the first A node. In this case, a child with edge labeled *1* already exists, so we use it rather than creating a new one. Rule 3 instructs us to label this child a, yielding (c). Now we expand the second A node. Again we choose rule 3. We are instructed to label the *1* child a, but it already has that label, so we do not need to do anything. Finally, in (d), the only remaining node is the bottom-most node, labeled a. Since its label is a terminal category, it does not need to be expanded, and we are done.

Let us back up to (c) again. Here we were free to choose rule 4 instead of rule 3 to expand the right-hand A node. Rule 4 instructs us to label the *1* child b, but we cannot, inasmuch as it is already labeled a. The derivation fails, and no dag is generated.

The language $L(G_2)$ is the set of dags produced by successful derivations, as shown in Figure 7. (The edges of the dags should actually be labeled with 1's and 2's, but I have suppressed the edge labels for the sake of perspicuity.)

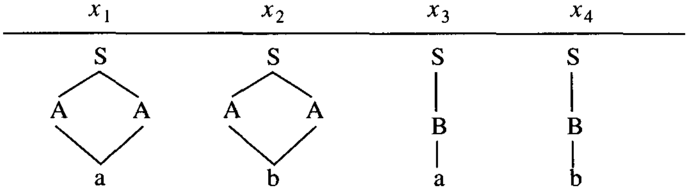


Figure 7
The language generated by G_2.

3.1 AV Grammars and the ERF Method

Now we face the question of how to attach probabilities to grammar G_2. The natural extension of the method we used for context-free grammars is the following: Associate a weight with each of the six rules of grammar G_2. For example, let M_2 be the model consisting of G_2 plus weights $(\beta_1, \ldots, \beta_6) = (1/2, 1/2, 2/3, 1/3, 1/2, 1/2)$. Let $\phi_2(x)$ be the weight that M_2 assigns to dag x; it is defined to be the product of the weights of the rules used to generate x. For example, the weight $\phi_2(x_1)$ assigned to tree x_1 of Figure 7 is 2/9, computed as in Figure 8. Rule 1 is used once and rule 3 is used twice; hence $\phi_2(x_1) = \beta_1\beta_3\beta_3 = 1/2 \cdot 2/3 \cdot 2/3 = 2/9$.

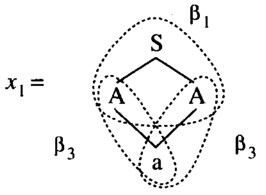


Figure 8
Rule applications in a dag generated by G_2. The weight of the dag is the product of the weights of rule applications.

Observe that $\phi_2(x_1) = \beta_1\beta_3^2$, which is to say, $\beta_1^{f_1(x_1)}\beta_3^{f_3(x_1)}$. Moreover, since $\beta^0 = 1$, it does not hurt to include additional factors $\beta_i^{f_i(x_1)}$ for those i where $f_i(x_1) = 0$. That is, we can define the dag weight ϕ corresponding to rule weights $\beta = (\beta_1, \ldots, \beta_n)$ generally as:

$$\phi(x) = \prod_{i=1}^{n} \beta_i^{f_i(x)}$$

The next question is how to estimate weights. Let us consider what happens when we use the ERF method. Let us assume a corpus distribution for the dags in Figure 7 analogous to the distribution in Figure 3:

$$\begin{array}{rcccc} & x_1 & x_2 & x_3 & x_4 \\ \tilde{p} = & 1/3 & 1/6 & 1/4 & 1/4 \end{array} \qquad (1)$$

Using the ERF method, we estimate rule weights as in Table 4. This table is identical to the one given earlier in the context-free case. We arrive at the same weights M_2 we considered above, defining dag weights $\phi_2(x)$.

Table 4
Estimating the parameters of G_2 using the ERF method.

	$\tilde{p}$	$\tilde{p}f_1$	$\tilde{p}f_2$	$\tilde{p}f_3$	$\tilde{p}f_4$	$\tilde{p}f_5$	$\tilde{p}f_6$
x_1	1/3	1/3		2/3			
x_2	1/6	1/6			2/6		
x_3	1/4		1/4			1/4	
x_4	1/4		1/4				1/4
	$\tilde{p}[f] =$	1/2	1/2	2/3	1/3	1/4	1/4
	$\beta =$	1/2	1/2	2/3	1/3	1/2	1/2

3.2 Why the ERF Method Fails

But at this point a problem arises: ϕ_2 is not a probability distribution. Unlike in the context-free case, the four dags in Figure 7 constitute the entirety of $L(G_2)$. This time, there are no missing dags to account for the missing probability mass.

There is an obvious "fix" for this problem: we can simply normalize ϕ_2. We might define the distribution q for an AV grammar with weight function ϕ as:

$$q(x) = \frac{1}{Z}\phi(x)$$

where Z is the normalizing constant:

$$Z = \sum_{x \in L(G)} \phi(x)$$

In particular, for ϕ_2, we have $Z = 2/9 + 1/18 + 1/4 + 1/4 = 7/9$. Dividing ϕ_2 by $7/9$ yields the ERF distribution:

$$\begin{array}{rcccc} & x_1 & x_2 & x_3 & x_4 \\ q_2(x) = & 2/7 & 1/14 & 9/28 & 9/28 \end{array}$$

On the face of it, then, we can transplant the methods we used in the context-free case to the AV case and nothing goes wrong. The only problem that arises (ϕ not summing to one) has an obvious fix (normalization).

However, something has actually gone very wrong. The ERF method yields the best weights only under certain conditions that we inadvertently violated by changing $L(G)$ and re-apportioning probability via normalization. In point of fact, we can easily see that the ERF weights in Table 4 are *not* the best weights for our example grammar. Consider the alternative model $M*$ given in Figure 9, defining probability distribution $q*$.

S → A A	S → B	A → a	A → b	B → a	B → b
$\frac{3+2\sqrt{2}}{6+2\sqrt{2}}$	$\frac{3}{6+2\sqrt{2}}$	$\frac{\sqrt{2}}{1+\sqrt{2}}$	$\frac{1}{1+\sqrt{2}}$	$\frac{1}{2}$	$\frac{1}{2}$

Figure 9
An alternative model, $M*$.

These weights are proper, in the sense that weights for rules with the same left-hand side sum to one. The reader can verify that $\phi*$ sums to $Z = \frac{3+\sqrt{2}}{3}$ and that $q*$ is:

$$\begin{array}{rcccc} & x_1 & x_2 & x_3 & x_4 \\ q*(x) = & 1/3 & 1/6 & 1/4 & 1/4 \end{array}$$

That is, $q* = \tilde{p}$. Comparing q_2 (the ERF distribution) and $q*$ to $\tilde{p}$, we observe that $D(\tilde{p}\|q_2) = 0.07$ but $D(\tilde{p}\|q*) = 0$.

In short, in the AV case, the ERF weights do *not* yield the best weights. This means that the ERF method does *not* converge to the correct weights as the corpus size increases. If there are genuine dependencies in the grammar, the ERF method converges systematically to the wrong weights. Fortunately, there are methods that do converge to the right weights. These are methods that have been developed for random fields.

4. Random Fields

A random field defines a probability distribution over a set of labeled graphs Ω called **configurations**. In our case, the configurations are the dags generated by the grammar, i.e., $\Omega = L(G)$. The weight assigned to a configuration is the product of the weights assigned to selected **features** of the configuration. We use the notation:

$$\phi(x) = \prod_i \beta_i^{f_i(x)}$$

where β_i is the weight for feature i and $f_i(\cdot)$ is its frequency function, that is, $f_i(x)$ is the number of times that feature i occurs in configuration x. (For most purposes, a feature can be identified with its frequency function; I will not always make a careful distinction between them.)

I use the term feature here as it is used in the machine learning and statistical pattern recognition literature, *not* as in the constraint grammar literature, where *feature* is synonymous with *attribute*. In my usage, dag edges are labeled with attributes, not features. Features are rather like geographic features of dags: a feature is some larger or smaller piece of structure that occurs—possibly at more than one place—in a dag.

The probability of a configuration (that is, a dag) is proportional to its weight, and is obtained by normalizing the weight distribution.

$$q(x) = \tfrac{1}{Z}\phi(x) \quad Z = \textstyle\sum_{x \in \Omega} \phi(x)$$

If we identify the features of a configuration with local trees—equivalently, with applications of rewrite rules—the random field model is almost identical to the model we considered in the previous section. There are two important differences. First, we no longer require weights to sum to one for rules with the same left-hand side. Second, the model does not *require* features to be identified with rewrite rules. We use the grammar to define the set of configurations $\Omega = L(G)$, but in defining a probability distribution over $L(G)$, we can choose features of dags however we wish.

Let us consider an example. Let us continue to assume grammar G_2 generating the language in Figure 7, and let us continue to assume the empirical distribution in (1). But now rather than taking rule applications to be features, let us adopt the two features in Figure 10. For purpose of illustration, take feature 1 to have weight $\beta_1 = \sqrt{2}$ and feature 2 to have weight $\beta_2 = 3/2$. The functions f_1 and f_2 represent the frequencies of features 1 and 2, respectively, as in Figure 11. We are able to exactly

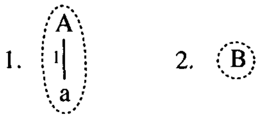
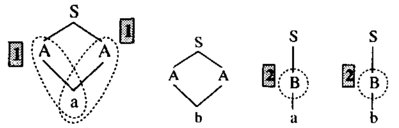


Figure 10
Two features.



$f_1 =$	2	0	0	0	
$f_2 =$	0	0	1	1	
$\phi =$	$\sqrt{2} \cdot \sqrt{2}$	1	3/2	3/2	$Z = 6$
$q =$	1/3	1/6	1/4	1/4	

Figure 11
The frequencies (number of instances) of features 1 and 2 in dags generated by G_2, and the computation of dag weights ϕ and dag probabilities q.

recreate the empirical distribution using fewer features than before. Intuitively, we need only use as many features as are necessary to distinguish among trees that have different empirical probabilities.

This added flexibility is welcome, but it does make parameter estimation more involved. Now we must not only choose values for weights, we must also choose the features that weights are to be associated with. We would like to do both in a way that permits us to find the best model, in the sense of the model that minimizes the Kullback-Leibler distance with respect to the empirical distribution. The IIS algorithm (Della Pietra, Della Pietra, and Lafferty 1995) provides a method to do precisely that.

5. Field Induction

In outline, the IIS algorithm is as follows:

1. Start ($t = 0$) with the null field, containing no features.
2. **Feature Selection.** Consider every feature that might be added to field M_t and choose the best one.
3. **Weight Adjustment.** Readjust weights for all features. The result is field M_{t+1}.
4. Iterate until the field cannot be improved.

For the sake of concreteness, let us take features to be labeled subdags. In step 2 of the algorithm we do not consider every conceivable labeled subdag, but only the atomic (i.e., single-node) subdags and those complex subdags that can be constructed by combining features already in the field or by combining a feature in the field with some atomic feature. We also limit our attention to features that actually occur in the training corpus.

In our running example, the atomic features are as shown in Figure 12. Features can be combined by adding connecting arcs, as shown in Figure 13, for example.

Figure 12
The atomic features arising in dags generated by G_2.

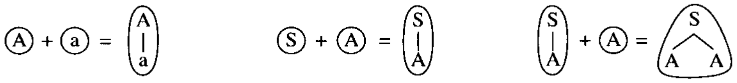
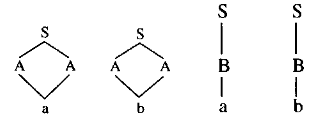


Figure 13
Combining features to create more complex features.

5.1 The Null Field

Field induction begins with the null field. With the corpus we have been assuming, the null field takes the form in Figure 14. No dag x has any features, so $\phi(x) = \prod_i \beta_i^{f_i(x)}$ is a



$$\phi(x) = \quad 1 \quad 1 \quad 1 \quad 1 \quad Z = 4$$
$$q(x) = \quad 1/4 \quad 1/4 \quad 1/4 \quad 1/4$$

Figure 14
The null field for G_2.

product of zero terms, and hence has value 1. As a result, q is the uniform distribution. The Kullback-Leibler divergence $D(\tilde{p}\|q)$ is 0.03. The aim of feature selection is to choose a feature that reduces this divergence as much as possible.

The astute reader will note that there is a problem with the null field if $L(G)$ is infinite. Namely, it is not possible to have a uniform probability mass distribution over an infinite set. If each dag in an infinite set of dags is assigned a constant nonzero probability ϵ, then the total probability is infinite, no matter how small ϵ is. There are a couple of ways of dealing with the problem. The approach that DD&L adopt is to assume a consistent prior distribution $p(k)$ over graph sizes k, and a family of random fields q_k representing the conditional probability $q(x \mid k)$; the probability of a tree is then $p(k)q(x \mid k)$. All the random fields have the same features and weights, differing only in their normalizing constants.

I will take a somewhat different approach here. As sketched at the beginning of section 3, we can generate dags from an AV grammar much as proposed by Brew and Eisele. If we ignore failed derivations, the process of dag generation is completely analogous to the process of tree generation from a stochastic CFG—indeed, in the limiting case in which none of the rules contain constraints, the grammar *is* a CFG. To obtain an initial distribution, we associate a weight with each rule, the weights for rules with a common left-hand side summing to one. The probability of a dag is proportional to the product of weights of rules used to generate it. (Renormalization is necessary because of the failed derivations.) We estimate weights using the ERF method: we estimate the weight of a rule as the relative frequency of the rule in the training corpus, among rules with the same left-hand side.

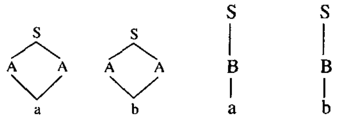
The resulting initial distribution (the ERF distribution) is not the maximum-likelihood distribution, as we know. But it can be taken as a useful first approximation. Intuitively, we begin with the ERF distribution and construct a random field to take account of context dependencies that the ERF distribution fails to capture, incrementally improving the fit to the empirical distribution.

In this framework, a model consists of: (1) An AV grammar G whose purpose is to define a set of dags $L(G)$. (2) A set of initial weights θ attached to the rules of G. The weight of a dag is the product of weights of rules used in generating it. Discarding failed derivations and renormalizing yields the initial distribution $p_0(x)$. (3) A set of features $f_1, \ldots, f_n$ with weights $\beta_1, \ldots, \beta_n$ to define the field distribution $q(x) = \frac{1}{Z} p_0(x) \prod_i \beta_i^{f_i(x)}$.

5.2 Feature Selection

At each iteration, we select a new feature f by considering all atomic features, and all complex features that can be constructed from features already in the field. Holding the weights constant for all old features in the field, we choose the best weight β for f (how β is chosen will be discussed shortly), yielding a new distribution $q_{\beta,f}$. The score for feature f is the reduction it permits in $D(\tilde{p}\|q_{\text{old}})$, where q_{old} is the old field. That is, the score for f is $D(\tilde{p}\|q_{\text{old}}) - D(\tilde{p}\|q_{\beta,f})$. We compute the score for each candidate feature and add to the field that feature with the highest score.

To illustrate, consider the two atomic features a and B. Given the null field as old field, the best weight for a is $\beta = 7/5$, and the best weight for B is $\beta = 1$. This yields q and $D(\tilde{p}\|f)$ as in Figure 15. The better feature is a, and a would be added to the field if these were the only two choices.



$\tilde{p}$	1/3	1/6	1/4	1/4	
ϕ_a	7/5	1	7/5	1	$Z = 24/5$
q_a	7/24	5/24	7/24	5/24	
$\tilde{p} \ln \frac{\tilde{p}}{q_a}$	0.04	−0.04	−0.04	0.05	$D = 0.01$
ϕ_B	1	1	1	1	$Z = 4$
q_B	1/4	1/4	1/4	1/4	
$\tilde{p} \ln \frac{\tilde{p}}{q_B}$	0.10	−0.07	0	0	$D = 0.03$

Figure 15
Comparing features. q_a is the best (minimum-divergence) distribution that can be generated by adding the feature "a" to the field, and q_B is the best distribution generable by adding the feature "B".

Intuitively, a is better than B because a permits us to distinguish the set $\{x_1, x_3\}$ from the set $\{x_2, x_4\}$; the empirical probability of the former is $1/3+1/4 = 7/12$ whereas the empirical probability of the latter is $5/12$. Distinguishing these sets permits us to model the empirical distribution better (since the old field assigns them equal probability, counter to the empirical distribution). By contrast, the feature B distinguishes the set $\{x_1, x_2\}$ from $\{x_3, x_4\}$. The empirical probability of the former is $1/3+1/6 = 1/2$ and the empirical probability of the latter is also $1/2$. The old field models these probabilities exactly correctly, so making the distinction does not permit us to improve on the old field. As a result, the best weight we can choose for B is 1, which is equivalent to not having the feature B at all.

5.3 Selecting the Initial Weight

DD&L show that there is a unique weight $\hat{\beta}$ that maximizes the score for a new feature f (provided that the score for f is not constant for all weights). Writing q_{β} for the distribution that results from assigning weight β to feature f, $\hat{\beta}$ is the solution to the equation

$$q_{\beta}[f] = \tilde{p}[f] \tag{2}$$

Intuitively, we choose the weight such that the expectation of f under the resulting new field is equal to its empirical expectation.

Solving equation (2) for β is easy if $L(G)$ is small enough to enumerate. Then the sum over $L(G)$ that is implicit in $q_{\beta}[f]$ can be expanded out, and solving for β is simply a matter of arithmetic. Things are a bit trickier if $L(G)$ is too large to enumerate. DD&L show that we can solve equation (2) if we can estimate $q_{\text{old}}[f = k]$ for k from 0 to the maximum value of f in the training corpus. (See Appendix 1 for details.)

We can estimate $q_{\text{old}}[f = k]$ by means of **random sampling**. The idea is actually rather simple: to estimate how often the feature appears in "the average dag," we generate a representative mini-corpus from the distribution q_{old} and count. That is, we generate dags at random in such a way that the relative frequency of dag x is $q_{\text{old}}(x)$ (in the limit), and we count how often the feature of interest appears in dags in our generated mini-corpus.

The application that DD&L consider is the induction of English orthographic constraints, that is, inducing a field that assigns high probability to "English-sounding" words and low probability to non-English-sounding words. For this application, Gibbs sampling is appropriate. Gibbs sampling does not work for the application to AV grammars, however. Fortunately, there is an alternative random sampling method we can use: Metropolis-Hastings sampling. We will discuss the issue in some detail shortly.

5.4 Readjusting Weights

When a new feature is added to the field, the best value for its initial weight is chosen, but the weights for the old features are held constant. In general, however, adding the new feature may make it necessary to readjust weights for all features. The second half of the IIS algorithm involves finding the best weights for a given set of features.

The method is very similar to the method for selecting the initial weight for a new feature. Let $(\beta_1, \ldots, \beta_n)$ be the old weights for the features. We wish to compute "increments" $(\delta_1, \ldots, \delta_n)$ to determine a new field with weights $(\delta_1\beta_1, \ldots, \delta_n\beta_n)$. Consider the equation

$$q_{\text{old}}[\delta_i^{f\#} f_i] = \tilde{p}[f_i] \tag{3}$$

where $f\#(x) = \sum_i f_i(x)$ is the total number of features of dag x. The reason for the factor $\delta_i^{f\#}$ is a bit involved. Very roughly, we would like to choose weights so that the expectation of f_i under the *new* field is equal to $\tilde{p}[f_i]$. Now $q_{\text{new}}(x)$ is:

$$\begin{aligned} q_{\text{new}}(x) &= \frac{1}{Z} p_0(x) \prod_j (\delta_j \beta_j)^{f_j(x)} \\ &= \frac{1}{Z_\delta} q_{\text{old}}(x) \prod_j \delta_j^{f_j x} \end{aligned}$$

where we factor Z as $Z_\delta Z_\beta$, for Z_β the normalization constant in q_{old}. Hence, $q_{\text{new}}[f_i] = q_{\text{old}}[\frac{1}{Z_\delta} f_i \prod_j \delta_j^{f_j x}]$. Now there are two problems with this expression: it requires us to compute Z_δ, which we are not able to do, and it requires us to determine weights

δ_j for all the features simultaneously, not just the weight δ_i for feature i. We might consider approximating $q_{\text{new}}[f_i]$ by ignoring the normalization factor and assuming that all features have the same weight as feature i. Since $\prod_j \delta_i^{f_j(x)} = \delta_i^{f_\#(x)}$, we arrive at the expression on the left-hand side of equation (3).

One might expect the approximation just described to be rather poor, but it is proven in Della Pietra, Della Pietra, and Lafferty (1995) that solving equation (3) for δ_i (for each i) and setting the new weight for feature i to $\delta_i\beta_i$ is guaranteed to improve the model. This is the real justification for equation (3), and the reader is referred to Della Pietra, Della Pietra, and Lafferty (1995) for details.

Solving (3) yields improved weights, but it does not necessarily immediately yield the globally best weights. We can obtain the globally best weights by iterating. Set $\beta_i \leftarrow \delta_i\beta_i$, for all i, and solve equation (3) again. Repeat until the weights no longer change.

As with equation (2), solving equation (3) is straightforward if $L(G)$ is small enough to enumerate, but not if $L(G)$ is large. In that case, we must use random sampling. We generate a representative mini-corpus and estimate expectations by counting in the mini-corpus. (See Appendix 2.)

5.5 Random Sampling

We have seen that random sampling is necessary both to set the initial weight for features under consideration and to adjust all weights after a new feature is adopted. Random sampling involves creating a corpus that is representative of a given model distribution $q(x)$. To take a very simple example, a fair coin can be seen as a method for sampling from the distribution q in which $q(H) = 1/2$, $q(T) = 1/2$. Saying that a corpus is representative is actually not a comment about the corpus itself but the method by which it was generated: a corpus representative of distribution q is one generated by a process that samples from q. Saying that a process M samples from q is to say that the empirical distributions of corpora generated by M converge to q in the limit. For example, if we flip a fair coin once, the resulting empirical distribution over (H, T) is either $(1, 0)$ or $(0, 1)$, not the fair-coin distribution $(1/2, 1/2)$. But as we take larger and larger corpora, the resulting empirical distributions converge to $(1/2, 1/2)$.

An advantage of SCFGs that random fields lack is the transparent relationship between an SCFG defining a distribution q and a sampler for q. We can sample from q by performing stochastic derivations: each time we have a choice among rules expanding a category X, we choose rule $X \rightarrow \xi_i$ with probability β_i, where β_i is the weight of rule $X \rightarrow \xi_i$.

Now we *can* sample from the initial distribution p_0 by performing stochastic derivations. At the beginning of Section 3, we sketched how to generate dags from an AV grammar G via nondeterministic derivations. We defined the initial distribution in terms of weights θ attached to the rules of G. We can convert the nondeterministic derivations discussed at the beginning of Section 3 into stochastic derivations by choosing rule $X \rightarrow \xi_i$ with probability θ_i when expanding a node labeled X. Some derivations fail, but throwing away failed derivations has the effect of renormalizing the weight function, so that we generate a dag x with probability $p_0(x)$, as desired.

The Metropolis-Hastings algorithm provides us with a means of converting the sampler for the initial distribution $p_0(x)$ into a sampler for the field distribution $q(x)$. Generally, let $p(\cdot)$ be a distribution for which we have a sampler. We wish to construct a sample $x_1, \ldots, x_N$ from a different distribution $q(\cdot)$. Assume that items $x_1, \ldots, x_n$ are already in the sample, and we wish to choose x_{n+1}. The sampler for $p(\cdot)$ *proposes* a new item y. We do not simply add y to the sample—that would give us a sample from $p(\cdot)$—but rather we make a stochastic decision whether to accept the proposal y or reject it. If we accept y, it is added to the sample ($x_{n+1} = y$), and if we reject y, then x_n is repeated ($x_{n+1} = x_n$).

The acceptance decision is made as follows: If $p(y) > q(y)$, then y is overrepresented among the proposals. We can quantify the degree of overrepresentation as $p(y)/q(y)$. The idea is to reject y with a probability corresponding to its degree of overrepresentation. However, we do not consider the absolute degree of overrepresentation, but rather the degree of overrepresentation relative to x_n. (If y and x_n are equally overrepresented, there is no reason to reject y in favor of x_n.) That is, we consider the value

$$r = \frac{p(y)/q(y)}{p(x_n)/q(x_n)} = \frac{p(y)q(x_n)}{p(x_n)q(y)}.$$

If $r \leq 1$, then y is underrepresented relative to x_n, and we accept y with probability one. If $r > 1$, then we accept y with a probability that diminishes as r increases: specifically, with probability $1/r$. In brief, the acceptance probability of y is $A(y \mid x_n) = \min(1, 1/r)$. It can be shown that proposing items with probability $p(\cdot)$ and accepting them with probability $A(\cdot \mid x_n)$ yields a sampler for $q(\cdot)$. (See, for example, Winkler [1995]).[2]

The acceptance probability $A(y \mid x_n)$ reduces in our case to a particularly simple form. If $r < 1$ then $A(y \mid x) = 1$. Otherwise, writing $\phi(x)$ for the "field weight" $\prod_i \beta_i^{f_i(x)}$, we have:

$$\begin{aligned} A(y \mid x_n) &= \frac{Z^{-1}\phi(y)p_0(y)p_0(x_n)}{Z^{-1}\phi(x_n)p_0(x_n)p_0(y)} \\ &= \phi(y)/\phi(x_n) \end{aligned} \qquad (4)$$

6. Final Remarks

In summary, we cannot simply transplant CF methods to the AV grammar case. In particular, the ERF method yields correct weights only for SCFGs, not for AV grammars. We can define a probabilistic version of AV grammars with a correct weight-selection method by going to random fields. Feature selection and weight adjustment can be accomplished using the IIS algorithm. In feature selection, we need to use random sampling to find the initial weight for a candidate feature, and in weight adjustment we need to use random sampling to solve the weight equation. The random sampling method that DD&L used is not appropriate for sets of dags, but we can solve that problem by using the Metropolis-Hastings method instead.

Open questions remain. First, random sampling is notorious for being slow, and it remains to be shown whether the approach proposed here will be practicable. I expect practicability to be quite sensitive to the choice of grammar—the more the grammar's

[2] The Metropolis-Hastings acceptance probability is usually given in the form

$$A(y \mid x) = \min\left(1, \frac{\pi(y)g(y,x)}{\pi(x)g(x,y)}\right)$$

in which π is the distribution we wish to sample from (q, in our notation) and $g(x, y)$ is the proposal probability: the probability that the input sampler will propose y if the previous configuration was x. The case we consider is a special case in which the proposal probability is independent of x: the proposal probability $g(x, y)$ is, in our notation, $p(y)$.

The original Metropolis algorithm is also a special case of the Metropolis-Hastings algorithm, in which the proposal probability is symmetric, that is, $g(x, y) = g(y, x)$. The acceptance function then reduces to $\min(1, \pi(y)/\pi(x))$, which is $\min(1, q(y)/q(x))$ in our notation. I mention this only to point out that it is a *different* special case. Our proposal probability is not symmetric, but rather independent of the previous configuration, and though our acceptance function reduces to a form (4) that is similar to the original Metropolis acceptance function, it is not the same: in general, $\phi(y)/\phi(x) \neq q(y)/q(x)$.

distribution diverges from the initial context-free approximation, the more features will be necessary to "correct" it, and the more random sampling will be called on.

A second issue is incomplete data. The approach described here assumes complete data (a parsed training corpus). Fortunately, an extension of the method to handle incomplete data (unparsed training corpora) is described in Riezler (1997), and I refer readers to that paper.

As a closing note, it should be pointed out explicitly that the random field techniques described here can be profitably applied to context-free grammars, as well. As Stanley Peters nicely put it, there is a distinction between *possibilistic* and *probabilistic* context-sensitivity. Even if the language described by the grammar of interest—that is, the set of possible trees—is context-free, there may well be context-sensitive statistical dependencies. Random fields can be readily applied to capture such statistical dependencies whether or not $L(G)$ is context-sensitive.

Appendix A: Initial Weight Estimation

In the feature selection step, we choose an initial weight β for each candidate feature f so as to maximize the *gain* $G = D(\tilde{p}\|q_{\text{old}}) - D(\tilde{p}\|q_{f,\beta})$ of adding f to the field. It is actually more convenient to consider log weights $\alpha = \ln \beta$. For a given feature f, the log weight $\hat{\alpha}$ that maximizes gain is the solution to the equation:

$$q_\alpha[f] = \tilde{p}[f]$$

where q_α is the distribution that results from adding f to the field with log weight α. This equation can be solved using Newton's method. Define

$$F(\alpha) = \tilde{p}[f] - q_\alpha[f] \tag{5}$$

To find the value of α for which $F(\alpha) = 0$, we begin at a convenient point α_0 (the "null" weight $\alpha_0 = 0$ recommends itself) and iteratively compute:

$$\alpha_{t+1} = \alpha_t - \frac{F(\alpha_t)}{F'(\alpha_t)} \tag{6}$$

Della Pietra, Della Pietra, and Lafferty (1995) show that $F'(\alpha_t)$ is equal to the negative of the variance of f under the new field, which I will write $-V_\alpha[f]$.

To compute the iteration (6) we need to be able to compute $F(\alpha_t)$ and $F'(\alpha_t)$. For $F(\alpha_t)$ we require $\tilde{p}[f]$ and $q_\alpha[f]$, and $F'(\alpha_t)$ can be expressed as $q_\alpha[f]^2 - q_\alpha[f^2]$. $\tilde{p}[f]$ is simply the average value of f in the training corpus. The remaining terms are all of the form $q_\alpha[f^r]$. We can re-express this expectation in terms of the old field q_{old}:

$$\begin{aligned} q_\alpha[f^r] &= \sum_x f^r(x) q_\alpha(x) \\ &= \frac{\sum_x f^r(x) e^{\alpha f(x)} q_{\text{old}}(x)}{\sum_x e^{\alpha f(x)} q_{\text{old}}(x)} \\ &= \frac{q_{\text{old}}[f^r e^{\alpha f}]}{q_{\text{old}}[e^{\alpha f}]} \end{aligned}$$

The expectations $q_{\text{old}}[f^r e^{\alpha f}]$ can be obtained by generating a random sample $(z_1, \ldots, z_N)$ of size N from q_{old} and computing the average value of $f^r e^{\alpha f}$. That is, $q_{\text{old}}[f^r e^{\alpha f}] \approx$

$(1/N)s_r(\alpha)$, where:

$$\begin{aligned} s_r(\alpha) &= \sum_k f^r(z_k)e^{\alpha f(z_k)} \\ &= \sum_u \text{count}_k[f(z_k) = u]u^r e^{\alpha u} \end{aligned}$$

This yields:

$$q_\alpha[f^r] = \frac{s_r(\alpha)}{s_0(\alpha)}$$

and the Newton iteration (6) reduces to:

$$\alpha_{t+1} = \alpha_t + \frac{s_0^2(\alpha_t)\tilde{p}[f] - s_0(\alpha_t)s_1(\alpha_t)}{s_0(\alpha_t)s_2(\alpha_t) - s_1(\alpha_t)^2}$$

To compare candidates, we also need to know the gain $D(\tilde{p}\|q_{\text{old}}) - D(\tilde{p}\|q_{\hat{\alpha}})$ for each candidate. This can be expressed as follows (Della Pietra, Della Pietra, and Lafferty 1995):

$$\begin{aligned} G(f, \hat{\alpha}) &= \tilde{p}[f]\ln\hat{\alpha} - \ln q_{\text{old}}[e^{\hat{\alpha}f}] \\ &\approx \tilde{p}[f]\ln\hat{\alpha} - \ln s_0(\hat{\alpha}) + \ln N \end{aligned}$$

Putting everything together, the algorithm for feature selection has the following form. The array $E[f]$ is assumed to have been initialized with the empirical expectations $\tilde{p}[f]$.

```
procedure SelectFeature () begin
    Fill array $C[f, u] = \text{count}_k[f(z_k) = u]$
    by sampling from old field
    $\hat{G} \leftarrow 0$, $g \leftarrow$ none
    for each $f$ in candidates do
        $\alpha \leftarrow 0$
        until $\alpha$ is accurate enough do
            $s_0 \leftarrow s_1 \leftarrow s_2 \leftarrow 0$
            for $u$ from 0 to $u_{\max}$ do
                $x \leftarrow C[f, u]e^{\alpha u}$
                $s_0 \leftarrow s_0 + x$
                $s_1 \leftarrow s_0 + xu$
                $s_2 \leftarrow s_0 + xu^2$
            end
            $\alpha \leftarrow \alpha + \frac{s_0^2 E[f] - s_0 s_1}{s_0 s_2 - s_1^2}$
        end
        $G \leftarrow \alpha E[f] - \ln s_0 + \ln N$
        if $G > \hat{G}$ then $\hat{G} \leftarrow G, g \leftarrow f, \hat{\alpha} \leftarrow \alpha$
    end
    return $g, \hat{\alpha}, \hat{G}$
end
```

Appendix B: Adjusting Field Weights

The procedure for adjusting field weights has much the same structure as the procedure for choosing initial weights. In terms of log weights, we wish to compute increments $(\delta_1, \ldots, \delta_n)$ such that the new field, with log weights $(\alpha_1 + \delta_1, \ldots, \alpha_n + \delta_n)$ has a lower divergence than the old field $(\alpha_1, \ldots, \alpha_n)$. We choose each δ_i as the solution to the equation:

$$\tilde{p}[f_i] = q_{\text{old}}[f_i e^{\delta_i f_\#}]$$

Again, we use Newton's method. We wish to find δ such that $F_i(\delta) = 0$, where:

$$F_i(\delta) = \tilde{p}[f_i] - q_{\text{old}}[f_i e^{\delta f_\#}]$$

As Della Pietra, Della Pietra, and Lafferty (1995) show, the first derivative is:

$$F_i'(\delta) = -q_{\text{old}}[f_i f_\# e^{\delta f_\#}]$$

We see that the expectations we need to compute by sampling from q_{old} are of form $q_{\text{old}}[f_i f_\#^r e^{\delta f_\#}]$. We generate a random sample $(z_1, \ldots, z_N)$ and define:

$$\begin{aligned} s_r(i,\delta) &= \sum_k f_i(z_k) f_\#(z_k)^r e^{\delta f_\#(z_k)} \\ &= \sum_m \sum_u \text{count}_k[f_i(z_k) = u \wedge f_\#(z_k) = m] u m^r e^{\delta m} \\ &= \sum_m m^r e^{\delta m} \sum_{k|f_\#(z_k)=m} f_i(z_k) \end{aligned}$$

As we generate the sample we update the array $C[i,m] = \sum_{k|f_\#(z_k)=m} f_i(z_k)$. We estimate $q_{\text{old}}[f_i f_\#^r e^{\delta f_\#}]$ as the average value of $f_i f_\#^r e^{\delta f_\#}$ in the sample, namely, $(1/N) s_r(i,\delta)$. This permits us to compute $F_i(\delta)$ and $F_i'(\delta)$. The resulting Newton iteration is:

$$\delta_{t+1} = \delta_t + \frac{N\tilde{p}[f_i] - s_0(i,\delta_t)}{s_1(i,\delta)}$$

The estimation procedure is:

```
procedure AdjustWeights $(\alpha_1, \ldots, \alpha_n)$ begin
    until the field converges do
        Fill array $C[i,m]$
        by sampling from $q_\alpha$
        for $i$ from 1 to $n$
            $\delta \leftarrow 0$
            until $\delta$ is sufficiently accurate do
            $s_0 \leftarrow s_1 \leftarrow 0$
                for $m$ from 0 to $m_{\max}$ do
                    $x \leftarrow C[i,m] e^{\delta m}$
                    $s_0 \leftarrow s_0 + x$
                    $s_1 \leftarrow s_1 + xm$
                end
            $\delta \leftarrow \delta + \frac{NE[f_i] - s_0}{s_1}$
            end
            $\alpha_i \leftarrow \alpha_i + \delta$
        end
    end
    return $(\alpha_1, \ldots, \alpha_n)$
end
```

Acknowledgments

This work has greatly profited from the comments, criticism, and suggestions of a number of people, including Yoav Freund, John Lafferty, Stanley Peters, Hans Uszkoreit, and members of the audience at talks I gave at Saarbrücken and Tübingen. Michael Miller and Kevin Mark introduced me to random fields as a way of dealing with context-sensitivities in language, planting the idea that led (much later) to this paper. Finally, I would especially like to thank Marc Light and Stefan Riezler for extended discussions of the issues addressed here and helpful criticism of my first attempts to present this material. All responsibility for flaws and errors of course remains with me.

References

Brew, Chris. 1995. Stochastic HPSG. In *Proceedings of EACL-95.*

Della Pietra, Stephen, Vincent Della Pietra, and John Lafferty. 1995. Inducing features of random fields. Technical Report CMU-CS-95-144, CMU.

Eisele, Andreas. 1994. Towards probabilistic extensions of constraint-based grammars. Technical Report Deliverable R1.2.B, DYANA-2.

Gibbs, W. 1902. *Elementary Principles of Statistical Mechanics.* Yale University Press, New Haven, CT.

Mark, Kevin, Michael Miller, Ulf Grenander, and Steve Abney. 1992. Parameter estimation for constrained context-free language models. In *Proceedings of the Fifth Darpa Workshop on Speech and Natural Language,* San Mateo, CA. Morgan Kaufman.

Riezler, Stefan. 1996. Quantitative constraint logic programming for weighted grammar applications. Talk given at LACL, September.

Riezler, Stefan. 1997. Probabilistic constraint logic programming. Arbeitspapiere des Sonderforschungsbereichs 340, Bericht Nr. 117, Universität Tübingen.

Winkler, Gerhard. 1995. *Image Analysis, Random Fields and Dynamic Monte Carlo Methods.* Springer.