Appears in *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-2001).*

Refining the Structure of a Stochastic Context–Free Grammar

Joseph Bockhorst[†‡]
joebock@cs.wisc.edu

Mark Craven[‡†]
craven@biostat.wisc.edu

[†]Department of Computer Sciences
University of Wisconsin
Madison, Wisconsin 53706

[‡]Department of Biostatistics & Medical Informatics
University of Wisconsin
Madison, Wisconsin 53706

Abstract

We present a machine learning algorithm for refining the structure of a stochastic context–free grammar (SCFG). This algorithm consists of a heuristic for identifying structural errors and an operator for fixing them. The heuristic identifies nonterminals in the model SCFG that appear to be performing the function of two or more nonterminals in the target SCFG, and the operator attempts to rectify this problem by introducing a new nonterminal. Structural refinement is important because most common SCFG learning methods set the probability parameters while leaving the structure of the grammar fixed. Thus, any structural errors introduced prior to training will persist. We present experiments that show our approach is able to significantly improve the accuracy of an SCFG designed to model an important class of RNA sequences called *terminators*.

1 Introduction

Stochastic context–free grammars (SCFGs) have long been used in the NLP community (where the are more commonly known as probabilistic context–free grammars) and recently have been applied to biological sequence analysis tasks, in particular RNA analysis. One typical learning problem is to create an SCFG from a set of unparsed training sequences that will accurately recognize previously unseen sequences in the class being modeled. The first step of this two step process is to create the underlying context–free grammar, which we refer to as the *structure* of the model. The second step, the assignment of the probability parameters, is usually performed through an application of an expectation maximization (EM) algorithm called the inside–outside algorithm [Lari & Young, 1990]. One of the limitations of this approach is that if the structure is incomplete or in error, there may be no assignment of probabilities that would result in an accurate model. This is the problem we address in this paper. We introduce a refinement step that can identify and fix a class of structural errors that occur when a nonterminal in the model grammar is performing the function of more than one nonterminal in the target. The particular location refined at each step is determined "diagnostically" by analyzing the interaction between the model and a training set. This method may be contrasted

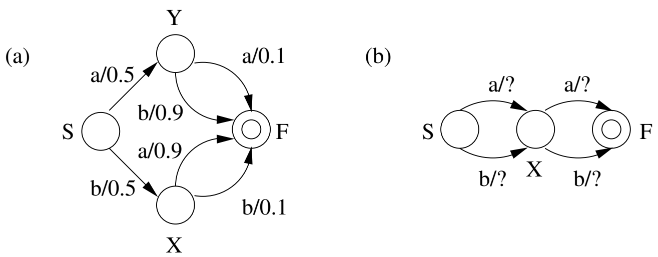

Figure 1: An example where an incorrect structure prevents the learning of an accurate model. (a) A target grammar to be learned. (b) The structure used to model sequences generated by the grammar shown in (a). State S is the begin state and F the end state. Arcs are labeled with the character emitted and the probability of taking that arc. The question marks in (b) represent the probability parameters to be learned. Because state X is overloaded in the learned model, it will be unable to correctly model the probabilities of the second character in the sequence.

to the standard state-space search methodology of applying the best of several competing operations as determined by a heuristic function applied to the successor states. This refinement step can be iterated with the inside–outside algorithm to refine both the structure of the model and the probability parameters. Through experiments with an SCFG designed to model a family of RNA sequences called *terminators* we show how this method may be used to learn more accurate models than inside–outside alone.

To see the impact a structural error can have, consider trying to learn the simple stochastic regular grammar shown in figure 1(a). Imagine that we have many sequences generated by this grammar for training and the structure we choose is shown in figure 1(b). Given enough training data, the maximum likelihood estimate of each state transition probability is 0.5. The probability of each length-two sequence under this model is therefore 0.25 resulting in a rather inaccurate model. The problem with this structure is that state X in the model is overloaded; it is performing the function of both state X and Y in the target. With the approach presented herein, errors such as this can potentially be corrected.

2 Problem Domain

The application of SCFGs we consider here is that of modeling of a class of RNA sequences called *terminators*. RNA sequences are strings from the alphabet {a,c,g,u} corresponding to the four different types of bases in the nucleotides of RNA molecules. The bases a and u are *complementary* to one another, as are c and g, because they are able to easily form *base pairs* by becoming chemically bonded[1].

The three dimensional shape that an RNA sequence assumes to a large degree determines its function, and the shape itself is strongly influenced by which particular bases are paired. For example, an element of RNA structure called a *stem* is formed when a number of adjacent bases pair with a complementary set of bases later in the sequence. If the intervening bases are unpaired, the resulting structure is called a *stem–loop*. Figure 2(a) shows an RNA sequence that could assume a stem–loop, and 2(b) shows the stem–loop it forms. The consequence of this is that families of RNA sequences of similar function will share a pattern of dependency between bases in the sequence that are likely to be paired.

Terminators are RNA sequences which signal when to stop the process of *transcription*, one of the key steps in the expression of a gene to form a protein.

3 Stochastic Context Free Grammars

The structure of an SCFG G is a context–free grammar and is defined by a set of nonterminals $\mathcal{N}$, a set of terminal symbols $\mathcal{T}$, a start nonterminal $S \in \mathcal{N}$ and a set of rewrite rules, or productions, of the form $N_v \rightarrow \Gamma, \Gamma \in \{(\mathcal{N} \backslash S) \cup \mathcal{T}\}^*$. That is, the right hand side of a production consists of any combination of terminal and nonterminal symbols other the start nonterminal. When we refer to N_v's productions, we mean the productions with N_v on the left hand side. Nonterminals and terminals are denoted by upper and lower case letters respectively. A grammar can be made into an SCFG by associating with each nonterminal N_v a probability distribution, $\mathbf{p_v}$, over N_v's productions.

SCFGs have proven important in RNA analysis tasks because the defining characteristics of RNA families manifest themselves in sequences through long range dependencies between bases. Such dependencies are troublesome to represent with regular grammars but can be naturally represented by context-free grammars. Figure 2(c) shows an SCFG for a specific set of stem–loops and Figure 2(d) shows the parse tree for an example sequence under this grammar.

There are three fundamental computational tasks relating SCFGs and sequences.

1. Compute $Pr(x^i|G)$, the probability of a sequence given a full model.
2. Find the most probable parse tree for a sequence.
3. Given G_s and a set of sequences $\mathcal{X} = \{x^1, x^2, ..., x^N\}$, set the probabilities to maximize the likelihood $\prod_{i=1}^{N} Pr(x^i|G)$.

Briefly, the first and second problems are efficiently solvable with dynamic programming algorithms similar to the forward and Viterbi algorithms, but there is no known efficient global solution for the third problem. The most common

[1]Base pairs can be formed between non-complementary bases but this is less common.

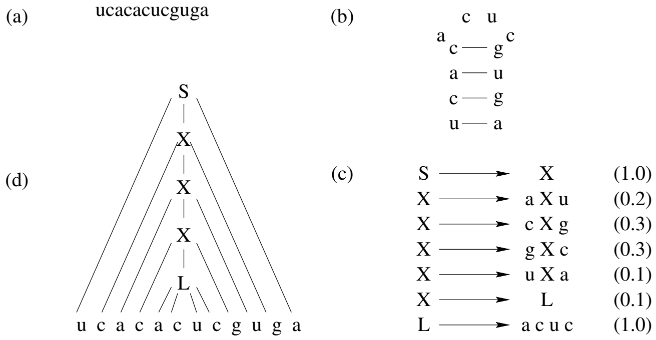

Figure 2: (a) A simple RNA sequence that will form a stem-loop structure. (b) The stem loop structure; paired bases are connected by dashes. (c) An SCFG model of sequences that form stem–loops. Probabilities are listed to the right of their associated production. (d) The parse tree for the sequence in (a) using the grammar in (c).

heuristic is the inside–outside [Lari & Young, 1990] method which, like other EM algorithms, converges to a local maximum.

The inside algorithm, used to solve task 1 above, is a dynamic programming algorithm that fills a three dimensional matrix α. If run on the sequence x^i, the elements of the matrix, $\alpha(j, k, v)$, contain the sum of the probabilities of all parse trees of the subsequence $x^i_j..x^i_k$ rooted at N_v. So, if the length of x^i is L and $N_1 = S$, then $\alpha(1, L, 1) = Pr(x^i|G)$.

The partner of the inside algorithm, the outside algorithm calculates a similar dynamic programming matrix β. An element $\beta(j, k, v)$ is the sum of the probabilities of all parse trees of the complete sequence x^i rooted at the start nonterminal, excluding all parse subtrees of $x^i..x^j$ rooted at N_v.

From α and β the expected number of times each nonterminal is used in the derivation of a sequence can be determined. The sum of these counts over all sequences in a training set are used by the inside–outside algorithm to iteratively re-estimate the production probabilities. In the next section we will show how α and β play a role in identifying overloaded nonterminals.

4 Grammar Refinement Approach

When the structure of a hypothesis grammar G^h is different from the structure of the target grammar $\mathcal{G}^*$, its accuracy may suffer. We would like to be able to identify and fix grammars in this situation using only a set of training sequences. In this section, we present a grammar refinement operator and a heuristic to do this. The operator is able to fix certain structural errors in G^h that occur when a nonterminal in the hypothesis is performing the function of more than one nonterminal of $\mathcal{G}^*$. We refer to such a nonterminal in G^h as *overloaded*. The heuristic identifies overloaded nonterminals by locating data dependencies that are not represented by the structure of G^h.

Before we delve into the details of the grammar refinement algorithm, it is helpful to consider the ways in which a hypothesis grammar may differ from the target grammar. Figure 3 gives a taxonomy of possible errors in a hypothesis grammar. The right branch out of the root contains all

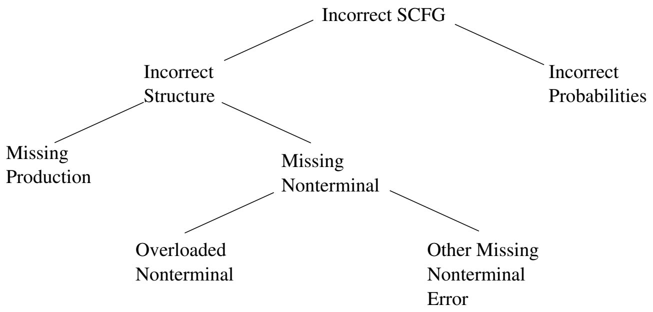

Figure 3: Possible inaccuracies of a hypothesis SCFG with respect to a target grammar $\mathcal{G}^*$

Table 1: The EXPAND operator. The operator creates a new nonterminal to take the place of nonterminal N on the right hand side of production P

EXPAND(P**,** N) /* N is on the RHS of production P */

1. Create new nonterminal N'
2. Replace N with N' in P
3. For each production $N \rightarrow \Gamma$, create $N' \rightarrow \Gamma$
4. Initialize probabilities for new productions

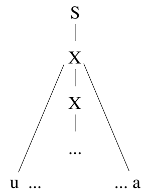

Figure 4: A partial parse tree where the nonterminal X is used in the two different contexts ($S \rightarrow X$, X) and ($X \rightarrow$a X u, X).

Table 2: The structure of the SCFG formed following an application of EXPAND($S \rightarrow X$, X) to the grammar in Figure 2(c). The new nonterminal X' replaces X in the production of the context being expanded, $S \rightarrow X$. Productions created by EXPAND are shown in boldface.

S → X'	
X → a X u	**X' → a X u**
X → c X g	**X' → c X g**
X → g X c	**X' → g X c**
X → a X u	**X' → u X a**
X → L	**X' → L**
L → a c u c	

SCFGs that have a correct structure; that is, there is some assignment of probabilities that results in a grammar equivalent to $\mathcal{G}^*$. Most learning algorithms for SCFGs, including inside–outside are aimed at this situation.

The left branch contains two categories of incorrectly structured grammars: those whose structures can be made correct by adding only productions and those that require additional nonterminals as well. Among the kinds of errors that can occur from missing nonterminals, the class we address occurs when a nonterminal in the hypothesis is overloaded because it is trying to perform the function of two or more nonterminals in the target. For example, the nonterminal X in the SCFG of Figure 2(c) would be overloaded if the distribution of the first base pair in the family of sequences being modeled was different from the rest. Next, we present a grammar refinement algorithm to identify and fix overloaded nonterminals.

The grammar refinement problem starts with set a sequences $\mathcal{X}$ which we can think of as generated by an unknown target SCFG $\mathcal{G}^*$, an initial hypothesis grammar, and a goal of discovering a model 'close' to $\mathcal{G}^*$. We navigate the space of grammar structures through the application of a grammar modification operator, applied at each step to a part of the current grammar chosen by a heuristic.

4.1 Refinement Operator

Our current grammar refinement procedure has a single operator, EXPAND, shown in Table 1, which is applied to what we call a *context*. We define a context to be a production and a nonterminal on the right hand side of that production. We denote a context either by the pair (P, N) where N is a nonterminal on the right hand side of production P or by c_{vr} where r indexes into the productions in which nonterminal N_v appears on the right hand side. Every nonterminal node in a parse tree defines a context. For example, the nonterminal X in the parse tree fragment of Figure 4 is in the context ($S \rightarrow X$, X) the first time it appears and ($X \rightarrow$a X u, X) the second. When applied to (P, N), EXPAND creates a new nonterminal N', replaces N with N' in P and creates a production with N' on the left hand side from each of N's productions. Table 2 shows the structure that is formed after applying EXPAND($S \rightarrow X, X$) to the SCFG of Figure 2(c). We explain how to set the probabilities of the new productions below.

4.2 Refinement Heuristics

The heuristic we use to guide the search expands a single context at each step. It chooses to expand the nonterminal that is used most differently in that context compared to its expected usage based on its probability distribution. Let $\mathbf{n_{vr}}$ be the vector of expected usage counts of the productions of N_v in context c_{vr} (i.e., on the right hand side of the production indicated by r) for the training sequences and let n_{vrl} be an element of this vector that refers to the expected number of times that the l^{th} production with N_v on the left hand side is used in context c_{vr}. EXPAND(c_{vr}) sets the probability distribution of the nonterminal it creates to the one defined by $\mathbf{n_{vr}}$. Given a measurement of difference between the data distribution n_{vr} and N_v's probability distribution $\mathbf{p_v}$ we define our heuristic to choose to expand the context that maximizes this difference. We have investigated two difference measures, one based on the Kullback–Leibler (KL) divergence and the other based on the χ^2 statistic.

The KL divergence, also called relative entropy, between two probability distributions $\mathbf{p}$ and $\mathbf{q}$ with the same discrete

Table 3: Example of values used in computing the heuristics for the context ($S \rightarrow X, X$) from the SCFG in Figure 2(c). The columns indicate: (1) the productions of X, (2) the observed number of times it is used in the context, (3) the observed probability distribution in the context, (4) the expected usage counts if the context's distribution were the same as the distribution over all contexts, (5) the distribution over all contexts.

	Context ($S \rightarrow X, X$)			All
Production	"Observed" Counts $\mathbf{n_{vr}}$	Pr $\mathbf{p_{vr}}$	Expected Counts $\mathbf{e_{vr}}$	Pr $\mathbf{p_v}$
$X \rightarrow$ a X u	5.0	0.10	10.0	0.2
$X \rightarrow$ c X g	26.5	0.53	15.0	0.3
$X \rightarrow$ g X c	10.0	0.20	15.0	0.3
$X \rightarrow$ u X a	3.5	0.07	5.0	0.1
$X \rightarrow L$	5.0	0.10	5.0	0.1
Sum	50.0	1.00	50.0	1.0

domain is defined as

$$KL(\mathbf{p}, \mathbf{q}) = \sum_i p_i log_2 \frac{p_i}{q_i}.$$

Consider the example in Table 3. Column (3) in this table shows $\mathbf{p_{vr}}$ for the productions of X in the context ($S \rightarrow X, X$). We can use KL divergence to compare this distribution to $\mathbf{p_v}$, shown in column (5), which is the probability distribution for the productions of X over all of X's contexts. Using this measure, we select the nonterminal N_v in context r that maximizes

$$KL(\mathbf{p_{vr}}, \mathbf{p_v}) \sum_l n_{vrl}$$

where $\mathbf{p_{vr}}$ is the probability distribution defined by $\mathbf{n_{vr}}$, and the sum calculates the expected total number of times N_v is in c_{vr}. By one interpretation, this measure selects the context that would waste the most bits in transmitting a message for each production applied in that context if the encoding used is the optimal encoding defined by $\mathbf{p_v}$ instead of $\mathbf{p_{vr}}$.

The second heuristic we consider, the χ^2 statistic, is commonly used to determine if the hypothesis that a data sample was drawn according to some distribution can be rejected. Using the vector of expected usage counts of nonterminal N_v in context c_{vr}, $\mathbf{n_{vr}}$, as the "observed" counts and $\mathbf{e_{vr}} = \mathbf{p_v} * (\sum_l n_{vrl})$ for the expected counts,

$$\chi^2_{vr} = \sum_l ((e_{vrl} - n_{vrl})^2 / e_{vrl}.$$

Our method selects for expansion the context c_{vr} which maximizes χ^2_{vr}. Returning to the example in Table 3, we can use χ^2 to assess the significance of differences between column (2), the observed counts for the productions applied in the given context, and column (4) the expected observations given the probabilities in column (5).

To apply either of these heuristics, we must first calculate the counts n_{vrl}. These can be calculated from the inside and outside matrices α and β in a similar way as the usage counts of productions and nonterminals are calculated in the inside–outside algorithm. For example, the expected number of times that the production $N_w \rightarrow$ c N_y g is used in the context ($N_v \rightarrow aN_wu, N_w$) in the derivation of the sequence x^i is

$$Pr(N_v \rightarrow aN_wu) Pr(N_w \rightarrow cN_yg) \sum_j \sum_k I\gamma_{jk}$$

where

$$\gamma_{jk} = \beta(j,k,v)\alpha(j+2,k-2,y)$$

and I is an indicator variable that is 1 if $x^i_j = a, x^i_{j+1} = c, x^i_{k-1} = g$ and $x^i_k = u$, and 0 otherwise. Also, the counts are smoothed by adding 1 to each n_{vrl}.

Table 4: The structure of our terminator grammar. Nonterminals are capitalized and the terminals are a, c, g and u. The notation X $\rightarrow$ Y | Z is shorthand for the two productions X $\rightarrow$ Y and X $\rightarrow$ Z. Productions containing t_l and t_r are shorthand for the family of 16 productions where t_l and t_r can be any of the four terminals. Productions containing t^*_l and t^*_r have a similar interpretation except that t^*_l and t^*_r can also be null allowing for unpaired bases in the interior of the stem.

START	$\rightarrow$	PREFIX STEM_BOT1 SUFFIX
PREFIX	$\rightarrow$	B B B B B B B B
STEM_BOT1	$\rightarrow$	t_l STEM_BOT2 t_r
STEM_BOT2	$\rightarrow$	t^*_l STEM_MID t^*_r | t^*_l STEM_TOP2 t^*_r
STEM_MID	$\rightarrow$	t^*_l STEM_MID t^*_r | t^*_l STEM_TOP2 t^*_r
STEM_TOP2	$\rightarrow$	t^*_l STEM_TOP1 t^*_r
STEM_TOP1	$\rightarrow$	t_l LOOP t_r
LOOP	$\rightarrow$	B B LOOP_MID
LOOP_MID	$\rightarrow$	B LOOP_MID | B B
SUFFIX	$\rightarrow$	B B B B B B B B
B	$\rightarrow$	a | c | g | u

5 Experiments

We have constructed an SCFG model of terminator sequences by incorporating known features of terminators identified in the terminator literature. The structure of the terminator grammar we start with, shown in Table 4, models the eight bases before and eight bases after the stem–loop and the stem–loop itself. The recursive nature of STEM_MID and LOOP_MID's productions enables variable length stems and loops to be represented. An unpaired base in the interior of the stem, a *bulge*, can also be represented.

Positive examples consist of the sequences of 142 known or proposed terminators of *E. coli*. Each of these sequences is 50 base pairs long with the 30^{th} base aligned to the assumed termination point. We have 125 negative sequences, also of length 50, which were collected from the regions following genes that are presumed to not contain terminators.

We performed a five fold cross validation experiment using the KL and χ^2 heuristic as well as a control where the context to EXPAND at each step is randomly chosen. The following steps were repeated.

1. Extract $\mathcal{T}$, the most probable subsequence of each positive sequence in the training set given the current model.

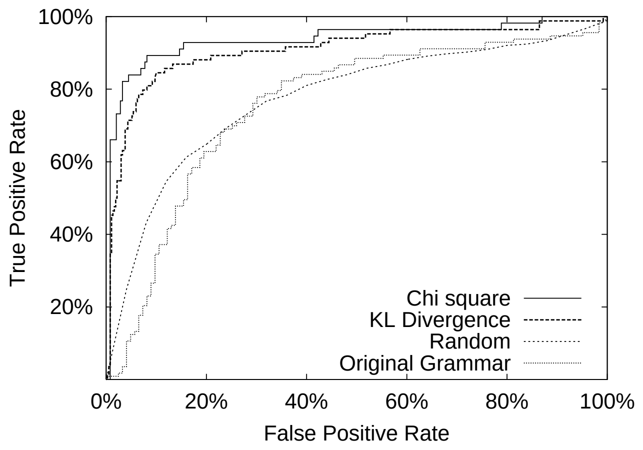

Figure 5: ROC curves of original grammar and the refined grammars after 25 additional nonterminals have been added using the χ^2, KL divergence and random heuristics. Note that the order of the models in the key reflects the order of the curves.

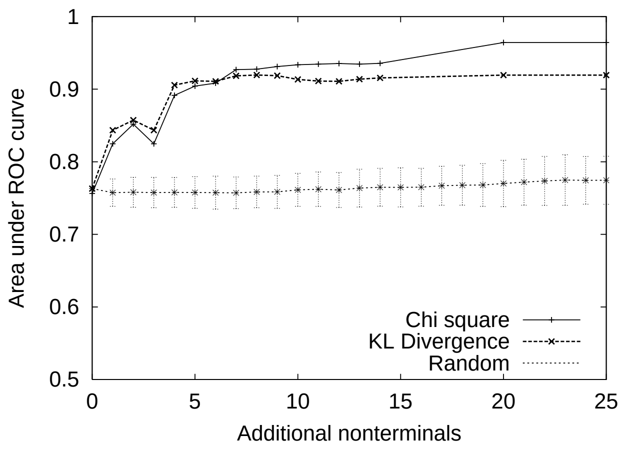

Figure 6: Plot of the area under the ROC curves for three heuristics: χ^2, KL Divergence, and random. The error bars on the random plot denote the 95% confidence interval. Note that the y-axis starts at 0.5 which corresponds to the expected area under the curve for random predictions.

2. Fit the probability parameters by running inside–outside to convergence on $\mathcal{T}$.
3. EXPAND the context chosen by the heuristic.

The first step is needed because the positive sequences contain more bases than what is being modeled by the grammar. The probabilities in the stem were initialized with a preference for complementary base pairs; elsewhere, a uniform distribution was used.

For each test sequence, we determine the probability of its most probable subsequence as given by each model. To compare various methods, we rank our test-set predictions by these probability values and then construct ROC curves. We get a single ROC curve for each approach by pooling predictions across test-sets.

Figure 5 shows the ROC curves that result from the original grammar, and from running our grammar refinement algorithm for 25 iterations using the χ^2, KL, and random heuristics. The random curve is the average over 35 random runs. The curves in this figure show that our heuristics provide better predictive accuracy than both the original grammar and our control of randomly selecting the context to expand at each step. These results support our hypothesis that directed changes to the structure of the grammar can result in more accurate learned models.

To consider how the predictive accuracy of the grammars changes as a function of the number of refinement iterations, we construct ROC curves after each iteration and then calculate the area under each curve. Figure 6 plots the area under these curves versus the number of nonterminals added for the three heuristics considered in Figure 5. The y-axis for this figure starts at 0.5 which is the expected area under the curve for a model that guessed randomly (such a model would result in an ROC "line" defined by: TP rate = FP rate). The results in this figure show that steady improvement is seen through about the 15th additional nonterminal using the KL heuristic and through about the 20th using χ^2. Our approach does not appear to overfit the training data, at least through the 25 iterations for which we have run it.

One possible reason that the χ^2 heuristic results in more accurate models than the KL heuristic is the following. We know that our calculations of KL Divergence are biased because they are calculated with finite samples [Herzel & Grosse, 1997], but that this bias is not uniform across the nonterminals and contexts we are comparing because the sample sizes differ. Therefore, one avenue for future research is to investigate measures that correct for the finite-sample bias of our KL Divergence calculation.

We have examined the sequence of contexts expanded for both the χ^2 and the KL Divergence heuristics, and noticed that several of the early refinements adjust a part of the grammar that describes the first few positions following the stem. There is known preference for t bases in this part of terminators, but our initial grammar did not encode it. This result suggests that our method can compensate for a known inadequacy in the initial grammar. However, we note that the algorithm makes additional modifications to other parts of the initial grammar that represent our best effort at encoding relevant domain knowledge.

One of the limitations of this algorithm as presented is the one-way nature of the search. That is, there is no way to make the grammar more general. If the initial grammar encodes the most general knowledge of the domain expert, this limitation may not be problematic initially. However, as EXPAND creates more nonterminals, it is likely that the grammar would become overly specific in places. The most obvious way of addressing this problem is by introducing a generalization operator. For example, a MERGE operator could be used to combine two nonterminals with the same RHS domain if their probability distributions are sufficiently similar. Note however, as mentioned by Stolcke (1994), that this operator will not introduce any new embedding structure into the grammar. This may not be a problem if such structure is present in the initial grammar.

6 Related Work

The approach we present here is related to the grammar induction algorithms of Stolcke (1994) and Chen (1996). These works both address the induction of SCFGs from text corpora for use as language models. Both methods incorporate a prior probability distribution over model structures and perform a search through posterior model probability space where Stolcke proceeds specific to general and Chen general to specific. One of the key differences between these algorithms and ours is that each of these addresses *tabula rasa* grammar induction while ours begins with a grammar structure created from prior knowledge. Another key difference is the way in which steps in the search space are selected. The methods of both Stolcke and Chen evaluate a candidate operator application by doing one-step lookahead. Since it can be expensive to calculate the posterior probability of the data given the changed model, heuristics are used to estimate this value. The operator application that results in the greatest estimated posterior probability is accepted. Our approach, on the other hand, is more "diagnostic." Instead of doing one-step lookahead, our heuristics try to directly identify places in which the grammar structure is inadequate. This approach may somewhat insulate our method from the overfitting pitfalls of likelihood driven searches. A third difference is that Stolcke and Chen use a richer set of operators than we do. As suggested in the previous section, however, we believe that our diagnostic approach can be generalized to work with other operators.

A different view of our approach is as an instance of a theory refinement algorithm [Pazzani & Kibler, 1992; Ourston & Mooney, 1994; Towell & Shavlik, 1994]. In theory refinement, the goal is to improve the accuracy of an incomplete or incorrect domain theory, from a set of labeled training examples. The primary difference between our work and previous work in theory refinement is the representation used by our learned models. Whereas previous theory-refinement methods have focused on logic-based and neural network representations, our learned models are represented using stochastic context free grammars.

The task we address is also similar to the problem of learning the structure of Bayesian networks [Heckerman, Geiger, & Chickering, 1995; Chickering, 1996], where the statistical properties of a training set are used to guide the modifications of the network structure. Again, the principal difference between our approach and this body of work is the difference in the representation language.

There has also been related work in learning SCFGs for RNA modeling tasks [Eddy & Durbin, 1994; Sakakibara *et al.*, 1994]. These iterative methods both update grammar structure using information gathered from the most probable parse of each of the training sequences at each step. Although we intend to compare our refinement algorithm to both of these approaches in future work, we consider our work to be complementary to these methods because it could be run on every one of their iterations.

7 Conclusion

We have considered the problem of refining the structure of a deficient SCFG using a set of training sequences. We introduced a grammar refinement operator, EXPAND, for repairing a type of SCFG structural error resulting from an overloaded nonterminal as well as a pair of heuristics, based on KL divergence and χ^2, for locating them. Preliminary results indicate that our method is able to improve the accuracy of an SCFG designed to model terminators by correcting known structural errors as well making modifications to parts of the grammar with no known deficiencies.

Acknowledgments

This research was supported in part by NSF CAREER award IIS-0093016 and NIH Grant 1R01 LM07050-01. Thanks to Soumya Ray for helpful comments on an earlier version of this paper.

References

[Chen, 1996] Chen, S. 1996. *Building Probabilistic Models for Natural Language*. Ph.D. Dissertation, Harvard University.

[Chickering, 1996] Chickering, D. M. 1996. Learning equivalence classes of Bayesian-network structure. In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA: Morgan Kaufmann.

[Eddy & Durbin, 1994] Eddy, S. R., and Durbin, R. 1994. RNA sequence analysis using covariance models. *Nucleic Acids Research* 22:2079–2088.

[Heckerman, Geiger, & Chickering, 1995] Heckerman, D.; Geiger, D.; and Chickering, D. M. 1995. Learning Bayesian networks. *Machine Learning* 20:197–243.

[Herzel & Grosse, 1997] Herzel, H., and Grosse, I. 1997. Correlations in DNA sequences: The role of protein coding segments. *Physical Review E* 55(1).

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

[Ourston & Mooney, 1994] Ourston, D., and Mooney, R. 1994. Theory refinement combining analytical and empirical methods. *Artificial Intelligence* 66(2):273–309.

[Pazzani & Kibler, 1992] Pazzani, M., and Kibler, D. 1992. The utility of knowledge in inductive learning. *Machine Learning* 9(1):57–94.

[Sakakibara *et al.*, 1994] Sakakibara, Y.; Brown, M.; Hughey, R.; Mian, I. S.; Sjölander, K.; Underwood, R. C.; and Haussler, D. 1994. Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Research* 22:5112–5120.

[Stolcke, 1994] Stolcke, A. 1994. *Bayesian Learning of Probabilistic Language Models*. Ph.D. Dissertation, University of California, Berkeley.

[Towell & Shavlik, 1994] Towell, G., and Shavlik, J. 1994. Knowledge-based artificial neural networks. *Artificial Intelligence* 70(1,2):119–165.