Stochastic Context-Free Grammars for Modeling RNA

Yasubumi Sakakibara[†], Michael Brown[†],
Rebecca C. Underwood[†], I. Saira Mian[§],
David Haussler[†]

UCSC-CRL-93-16
June 8, 1993

† Computer and Information Sciences
§ Sinsheimer Laboratories
University of California, Santa Cruz, CA 95064, USA.

email: haussler@cse.ucsc.edu

ABSTRACT

Stochastic context-free grammars (SCFGs) are applied to the problems of folding, aligning and modeling families of homologous RNA sequences. These models capture the common primary and secondary structure of the sequences with a context-free grammar, much like those used to define the syntax of programming languages. SCFGs generalize the hidden Markov models used in related work on protein and DNA sequences. The novel aspect of this work is that the SCFGs developed here are learned automatically from initially unaligned and unfolded training sequences. To do this, a new generalization of the forward-backward algorithm, commonly used to train hidden Markov models, is introduced. This algorithm is based on tree grammars, and is more efficient than the inside-outside algorithm, which was previously proposed to train SCFGs. This method is tested on the family of transfer RNA (tRNA) sequences. The results show that the model is able to reliably discriminate tRNA sequences from other RNA sequences of similar length, that it can reliably determine the secondary structure of new tRNA sequences, and that it can produce accurate multiple alignments of large collections of tRNA sequences. The model is also extended to handle introns present in tRNA genes.

Keywords: Stochastic Context-Free Grammar, RNA, Transfer RNA, Multiple Sequence Alignments, Database Searching.

1 Introduction

Attempts to understand the folding, structure, function and evolution of molecules has resulted in the confluence of many diverse disciplines ranging from structural biology and chemistry, through computer science and computational linguistics. Rapid generation of sequence data in recent years thus provides abundant opportunities for developing of new approaches to problems in computational biology such as Hidden Markov Models (HMMs) [Rab89, HKMS93, KBM+92, BCHM93, CS92]. In this paper, we apply *stochastic context-free grammars* (SCFGs) to the problems of statistical modeling, database searching, multiple alignment, and prediction of the secondary structure of RNA families. This approach is highly related to our previous work on modeling protein families with HMMs [HKMS93, KBM+92].

RNA is mostly involved in the biological machinery that expresses the genetic information from DNA to protein. Information is encoded in RNA by the linear arrangement of the four different constituent nucleotides (the primary structure). The individual nucleotides, adenine (`A`), cytosine (`C`), guanine (`G`) and uracil (`U`), interact in specific ways to form characteristic secondary structure motifs such as helices, loops and bulges. Further folding and hydrogen-bonding interactions between remote regions orient these secondary structure elements with respect to each other to form the functional system. Higher order interactions with other proteins and/or nucleic acids may also occur. In general, however, the folding of an RNA chain into a functional molecule is largely governed by the formation of intramolecular `A-U` and `G-C` Watson-Crick pairs as well as `G-U` base pairs.

Comparative analyses of two or more protein or nucleic acid sequences have been used widely in the detection and evaluation of biological similarities and evolutionary relationships. Several methods of producing these multiple sequence alignments have been developed, most based on dynamic programming techniques (see for example [Wat89]). However, when RNA sequences are to be aligned, both the primary and *secondary* structure need to be taken into consideration since the generation of a multiple sequence alignment and an analysis of folding are not mutually exclusive exercises. Thus, the elucidation of common folding patterns among two or more sequences may indicate the pertinent regions to be aligned and *vice versa* [San85].

Currently, there are two principal methods for predicting the secondary structure of RNA. Phylogenetic analysis for homologous RNA molecules relies upon alignment and subsequent folding of several sequences into similar structures (reviewed in [JOP89, WGGN83]). In contrast, energy minimization is dependent upon thermodynamic parameters and computer algorithms to evaluate the optimal and suboptimal free energy folding of an RNA species (reviewed in [JTZ90, ZS84]).

Our method of multiple alignment and folding differs quite markedly from the conventional techniques. Essentially, our method builds a statistical model during the process of multiple alignment and folding analysis, rather than leaving this as a separate task to be done after the alignment and folding are completed. In our previous studies [HKMS93, KBM+92], this approach has been successfully applied to modeling protein families with HMMs.

Although in principle HMMs could be used for RNA, we strongly suspect that the more general statistical models described below will be required to obtain useful results. Since base pairing interactions, most notably `A-U`, `G-C` and `G-U`, play such a dominant role in determining RNA structure and function, any statistical method that does not consider this will eventually encounter insurmountable problems. The problem is that if two positions are base paired in the typical RNA,

then the bases occurring at these two positions will be highly correlated, whereas they are treated as having independent distributions in the standard HMM approach. Such base pairs constitute so-called *biological palindromes* in the genome. We have found a way to generalize HMMs to model most of these interactions seen in RNA.

The essence of the idea can be expressed most clearly in terms of formal language theory. As in the work of Searls [Sea92], we can view the strings of characters representing pieces of DNA, RNA and protein as sentences derived from a formal grammar. The simplest kind of grammar is a *regular* grammar, in which strings are derived from productions (rewriting rules) of the form $S \rightarrow aS$ or $S \rightarrow a$, where S is a *nonterminal symbol* that does not appear in the final string, and a is a *terminal symbol*, which will appear as a letter in the final string. Searls has shown base pairing in RNA can be described by a *context-free grammar* (CFG), a more powerful class of formal grammars than the regular grammar (see Section 2.1 for an example). A CFG is similar to a regular grammar but permits a greater range of productions, such as those of the form $S \rightarrow SS$ and $S \rightarrow aSa$. As is beautifully described by Searls, it is precisely these additional types of production that are needed to describe the base pairing structure in RNA[1] [Sea92]. In particular, the productions of the forms $S \rightarrow \mathtt{A}\ S\ \mathtt{U}$, $S \rightarrow \mathtt{U}\ S\ \mathtt{A}$, $S \rightarrow \mathtt{G}\ S\ \mathtt{C}$, and $S \rightarrow \mathtt{C}\ S\ \mathtt{G}$ describe the structure in RNA due to Watson-Crick base pairing. Using productions of this type, a CFG can specify the language of biological palindromes.

If we specify a probability for each production in a grammar, we obtain a *stochastic grammar*. A stochastic grammar assigns a probability to each string it derives. Stochastic regular grammars are exactly equivalent to HMMs. This provides an alternate way of examining HMMs and suggests an interesting generalization from HMMs to *stochastic context-free grammars* (SCFGs) [Bak79].

In this paper, we pursue a stochastic model of the family of transfer RNAs (tRNAs) by using a SCFG that is similar to our previous protein HMMs [KBM+92] but which additionally incorporates base pairing information. A SCFG that forms a statistical model of tRNA sequences can be built in much the same way as our construction of an HMM representing a statistical model of the globin protein family. We use such a model to search a database for tRNA-like sequences and to obtain a multiple alignment in the same manner as for globins. We also use the model to fold unfolded tRNA sequences, i.e., to determine the base pairing that defines their secondary structure.

First, in order to see how well the SCFG can model families of RNA sequences, especially their common primary and secondary structure, we derive a SCFG directly from an existing alignment of tRNA sequences. We then repeat this experiment, but this time we attempt to "learn" the parameters entirely automatically from a set of *unaligned* primary sequences. To do this, we introduce a new generalization of the *forward-backward algorithm*, commonly used to train HMMs. Our algorithm is based on tree grammars, and is more efficient than the *inside-outside algorithm* [LY90], a computationally expensive generalization of the forward-backward algorithm developed by J. K. Baker to train SCFGs [Bak79]. Thus we derive two grammars: the *alignment grammar*, directly derived from an existing multiple alignment of tRNAs, and the *trained grammar*, deduced by our training algorithm from a training set of tRNA sequences. For our training set, we chose 500 sequences at random from 1477 tRNA sequences in EMBL Data Library's database. These

[1] Not all RNA structure can be described by CFGs but we believe they can account for enough to make useful models. In particular, CFGs cannot account for pseudoknots, structures generated when a single-stranded loop region forms standard Watson-Crick base pairs with a complementary sequence outside the loop.

training sequences are unfolded and unaligned. We withhold the remaining 977 sequences in order to test the trained grammar on data not used in the training process.

We compare the two grammars by evaluating their abilities to perform three tasks: to discriminate tRNA sequences from non-tRNA sequences, to produce multiple alignments, and to ascertain the secondary structure of new sequences. The results show that both grammars can perfectly discriminate tRNA sequences from other RNA sequences of similar length, can produce accurate multiple alignments of large collections of tRNA sequences, and can reliably determine the secondary structure of new tRNA sequences.

Surprisingly, the trained grammar can discriminate more reliably than the alignment grammar because the trained grammar exhibits a greater gap between Z-scores of tRNAs and non-tRNAs. This is unexpected because the trained grammar is obtained using only 500 tRNA training sequences, while the alignment grammar is obtained using all 1477 aligned tRNA sequences (including folding information).

Genes for tRNA often possess introns, regions that are excised out during formation of the mature tRNA molecule, i.e., the DNA sequence coding for a particular tRNA contains additional nucleotides that are not present in the RNA that folds to form the final structure. This means that when we search databank files that represent genomic sequences (such as those in GenBank), the grammar needs to be extended to handle this situation in order to correctly identify tRNAs. A useful advantage of SCFGs is that an intron grammar can be deduced separately from the plain tRNA grammar and these two separate grammars can then be combined into a single grammar. In a preliminary experiment, we use 55 sequences of introns for training a (sub)grammar to model introns, and combine two trained grammars for introns and intron-free tRNAs into a single grammar modeling tRNAs with introns. We test the grammar on the same 55 tRNA sequences with introns, and the grammar correctly identifies the positions of introns and the introns themselves in 80% of these sequences. Further work, using separate training and testing sets of larger size, is underway.

2 Methods

2.1 Context-free grammars as models of RNA

The context-free grammar (CFG) is a more powerful class of formal grammars than the regular grammar and is often used to define the syntax of programming languages. An example CFG that generates a particular set of RNA sequences is shown in Figure 2.1. We will use it to describe CFGs. See Searls [Sea92] for a more comprehensive explanation.

A formal grammar is a set of productions (rewriting rules) that are used to generate a set of strings, that is, a *language*. The productions are applied iteratively to generate a string, a process called *derivation*. For example, the grammar in Figure 2.1 generates the RNA sequence `CAUCAGGGAAGAUCUCUUG` by the following derivation: Beginning with the start symbol S_0, any production with S_0 left of the arrow can be chosen to replace S_0. If the production $S_0 \rightarrow S_1$ is selected (in this case, this is the only production available), the effect is to replace S_0 with the symbol S_1. This one derivation step is written $S_0 \Rightarrow S_1$, where the double arrow signifies application of a production. Next, if the production $S_1 \rightarrow$ `C` S_2 `G` is selected, the derivation step is $S_1 \Rightarrow$ `C` S_2 `G`. Continuing with similar derivation operations, each time choosing a nonterminal

$$
\begin{array}{lll}
\text{Productions } P = \{ & S_0 \to S_1, & S_7 \to \mathtt{G}\ S_8, \\
& S_1 \to \mathtt{C}\ S_2\ \mathtt{G}, & S_8 \to \mathtt{G}, \\
& S_1 \to \mathtt{A}\ S_2\ \mathtt{U}, & S_8 \to \mathtt{U}, \\
& S_2 \to \mathtt{A}\ S_3\ \mathtt{U}, & S_9 \to \mathtt{A}\ S_{10}\ \mathtt{U}, \\
& S_3 \to S_4\ \ S_9, & S_{10} \to \mathtt{C}\ S_{10}\ \mathtt{G}, \\
& S_4 \to \mathtt{U}\ S_5\ \mathtt{A}, & S_{10} \to \mathtt{G}\ S_{11}\ \mathtt{C}, \\
& S_5 \to \mathtt{C}\ S_6\ \mathtt{G}, & S_{11} \to \mathtt{A}\ S_{12}\ \mathtt{U}, \\
& S_6 \to \mathtt{A}\ S_7, & S_{12} \to \mathtt{U}\ S_{13}, \\
& S_7 \to \mathtt{U}\ S_7, & S_{13} \to \mathtt{C} \qquad \}
\end{array}
$$

Figure 2.1: This set of productions generates RNA sequences with a certain restricted structure. An example of this structure is shown in Figure 2.2. The symbols $S_0, S_1, \ldots, S_{13}$ are nonterminals and `A, U, G, C` are terminals representing the four nucleotides.

symbol and replacing it with the right hand side of an appropriate production, we obtain the following derivation terminating with the desired sequence:

$$
\begin{aligned}
S_0 &\Rightarrow S_1 \Rightarrow \mathtt{C}S_2\mathtt{G} \Rightarrow \mathtt{CA}S_3\mathtt{UG} \Rightarrow \mathtt{CA}S_4S_9\mathtt{UG} \Rightarrow \mathtt{CAU}S_5\mathtt{A}S_9\mathtt{UG} \Rightarrow \mathtt{CAUC}S_6\mathtt{GA}S_9\mathtt{UG} \\
&\Rightarrow \mathtt{CAUCA}S_7\mathtt{GA}S_9\mathtt{UG} \Rightarrow \mathtt{CAUCAG}S_8\mathtt{GA}S_9\mathtt{UG} \Rightarrow \mathtt{CAUCAGGGA}S_9\mathtt{UG} \Rightarrow \mathtt{CAUCAGGGAA}S_{10}\mathtt{UUG} \\
&\Rightarrow \mathtt{CAUCAGGGAAG}S_{11}\mathtt{CUUG} \Rightarrow \mathtt{CAUCAGGGAAGA}S_{12}\mathtt{UCUUG} \Rightarrow \mathtt{CAUCAGGGAAGAU}S_{13}\mathtt{UCUUG} \\
&\Rightarrow \mathtt{CAUCAGGGAAGAUCUCUUG}.
\end{aligned}
$$

Formally, a *context-free grammar* consists of a set of nonterminal symbols N, a terminal alphabet Σ, a set P of productions (rewriting rules), and the start symbol S_0. For a nonempty set X of symbols, let X^* denote the set of all finite strings of symbols in X. Every CFG production has the form $S \to \alpha$ where $S \in N$ and $\alpha \in (N \cup \Sigma)^*$. That is, the left-hand side consists of one nonterminal and there is no restriction on the number or placement of nonterminals and terminals on the right-hand side. The production $S \to \alpha$ means that the nonterminal S can be replaced by the string α. If $S \to \alpha$ is a production of P, then for any strings γ and δ in $(N \cup \Sigma)^*$, we define $\gamma S \delta \Rightarrow \gamma\alpha\delta$ and we say that $\gamma S \delta$ *directly derives* $\gamma\alpha\delta$ in G. We say β can be *derived* from α, denoted $\alpha \stackrel{*}{\Rightarrow} \beta$, if there exists a sequence of direct derivations $\alpha_0 \Rightarrow \alpha_1$, $\alpha_1 \Rightarrow \alpha_2$, ... , $\alpha_{n-1} \Rightarrow \alpha_n$ where $\alpha_0 = \alpha$, $\alpha_n = \beta$, $\alpha_i \in (N \cup \Sigma)^*$, and $n \geq 0$. Such a sequence is called a *derivation*. Thus a derivation corresponds to an order of applying productions to generate a string. The language *generated by* the grammar is the set of all terminal strings w that can be derived from the grammar, that is, the language $\{w \in \Sigma^* \mid S_0 \stackrel{*}{\Rightarrow} w\}$.

Our work in modeling RNA uses only productions of the following forms: $S \to SS$, $S \to aSa$, $S \to aS$, $S \to S$, or $S \to a$, where S is a nonterminal and a is a terminal. Productions may have one of the following forms: $S \to aSa$, used to describe the base-pairing in RNA; $S \to aS$ and $S \to a$, used to describe a loop of unpaired bases; $S \to SS$, used to describe the branched secondary structure; and $S \to S$, (called *skip productions*), used in the context of multiple alignments, as described below.

A derivation can be arranged in a tree structure, called a *parse tree*. A parse tree represents the syntactic structure of an RNA sequence given by the grammar, and hence reflects the actual

physical secondary structure. Figure 2.2 shows the derivation arranged in a parse tree reflecting the physical secondary structure.

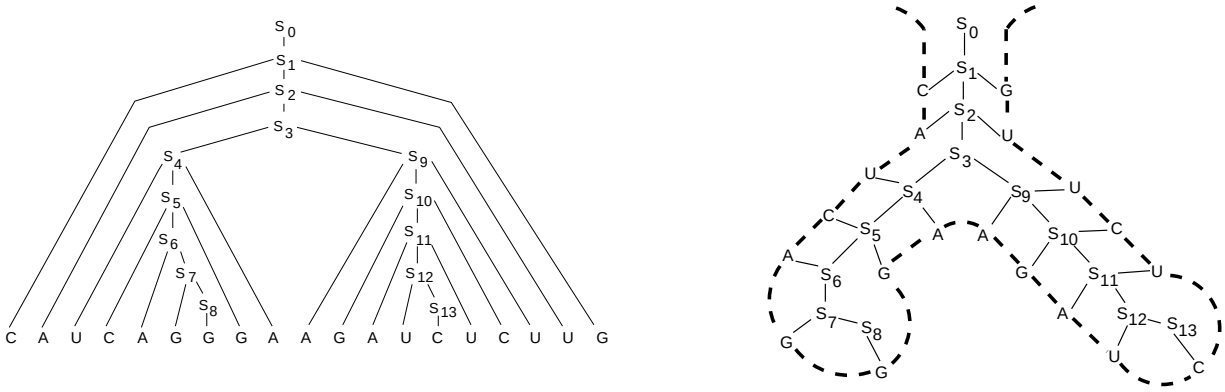


Figure 2.2: For the RNA sequence CAUCAGGGAAGAUCUCUUG, the grammar depicted in Figure 2.1 gives a parse tree (left) that reflects corresponding secondary structure (right).

As in the HMM of [KBM+92], we distinguish two different types of nonterminals: *match nonterminals* and *insert nonterminals*. The match nonterminals in a grammar correspond to important structural positions in an RNA or columns in a multiple alignment with few - (gap) characters. These constitute the *main line* of the grammar. Insert nonterminals generate nucleotides in exactly the same way as the match nonterminals, but use different distributions. These are used to insert extra nucleotides between important positions that correspond to columns in a multiple alignment. Skip productions are used to skip a match nonterminal, so that no nucleotide appears at that position in a multiple alignment.

2.2 Stochastic context-free grammars

In a *stochastic* context-free grammar (SCFG), every production for a nonterminal S has an associated probability value, such that a probability distribution exists over the set of productions for S. We denote the associated probability for a production $S \rightarrow \alpha$ by $\mathcal{P}(S \rightarrow \alpha)$. (Any production with the nonterminal S on the left-hand side is called "a production for S.")

An SCFG, G, generates sequences and assigns a probability to each generated sequence, and hence defines a probability distribution on the set of sequences. The probability of a parse tree can be calculated as the product of the probabilities of the productions used to generate the sequence. The probability of a sequence s is the sum of probabilities over all possible parse trees or derivations that could generate s, written as follows:

$$\begin{aligned}
\text{Prob}(s \mid G) &= \sum_{\text{all derivations (or parse trees) } d} \text{Prob}(S_0 \stackrel{d}{\Rightarrow} s \mid G) \\
&= \sum_{\alpha_1, \ \ldots \ ,\alpha_n} \text{Prob}(S_0 \Rightarrow \alpha_1 \mid G) \cdot \text{Prob}(\alpha_1 \Rightarrow \alpha_2 \mid G) \cdot \ \cdots \ \cdot \text{Prob}(\alpha_n \Rightarrow s \mid G)
\end{aligned}$$

Efficiently computing this quantity, $\text{Prob}(s \mid G)$, presents a problem because the number of possible parse trees for s is exponential in the length of the sequence. However, a dynamic programming technique analogous to the Cocke-Kasami-Young or Early methods [AU72] for non-stochastic CFGs can accomplish this task efficiently (in time proportional to the cube of the length of s). We define the negative logarithm of the probability of a sequence given by the grammar, i.e., $-\log(\text{Prob}(s \mid G))$, as the *negative log likelihood (NLL)-score* of the sequence. This quantifies how well the sequence s fits the grammar.

Since CFGs generally have an ambiguity in that the grammar gives more than one parse tree for a sequence, and alternative parse trees reflect alternative secondary structures (foldings), a grammar often gives several possible secondary structures for one RNA sequence. An advantage of a SCFG is that it can provide the most likely parse tree from this set of possibilities. If the grammar and the probabilities are carefully designed, the correct secondary structure will appear as the most likely parse tree among the alternatives. As discussed in Section 3.3, the most likely parse tree given by the trained grammar we produce for tRNAs gives exactly the correct secondary structures for the tRNA sequences we test.

We can compute the most likely parse tree efficiently using a variant of the above procedure for calculating $\text{Prob}(s \mid G)$. To obtain the most likely parse tree for the sequence s, we calculate

$$\max_{\text{parse trees } d} \text{Prob}(S_0 \stackrel{d}{\Rightarrow} s \mid G).$$

The dynamic programming procedure to do this resembles the Viterbi algorithm for HMMs [Rab89]. We can also use this procedure to obtain our multiple alignments: the grammar aligns each sequence by finding the most likely parse tree, after which the mutual alignment of the sequences among themselves is determined.

2.3 Estimating SCFGs from sequences

Searls [Sea92] argues the benefits of using context-free grammars as models for RNA folding, but does not discuss methods for estimating the grammar from training sequences. One purpose of this paper is to provide an effective method for estimating a SCFG to model a family of RNA sequences.

SCFGs from multiple alignments

All parameters in the SCFG (i.e., the production probabilities) could in principle be chosen from an existing alignment of RNA sequences. The method that we use to derive a SCFG from a multiple alignment estimates a distribution of four nucleotides for each column in the alignment corresponding to a nucleotide that is not base paired, and a distribution of 16 pairs of nucleotides for each pair of columns corresponding to nucleotides that are base paired in the secondary structure. Essentially, this is done by counting the occurrences of each letter in a column. However, our method uses Dirichlet mixture density priors for interpreting this count data to avoid statistical problems that arise when not enough count data are available. Details of this general method are described in [BHK+93].

EM training algorithm

In order to estimate the parameters of a SCFG from unaligned training RNA sequences, we introduce a new method for training SCFGs that is a generalization of the forward-backward algorithm, commonly used to train HMMs. This algorithm is more efficient than the inside-outside algorithm, which was previously proposed to train SCFGs.

The inside-outside algorithm [LY90, Bak79] is an Expectation Maximization (EM) algorithm to reestimate the parameters (i.e., the probabilities of productions) in a SCFG. However, it has some drawbacks when applied to practical problems: it requires the grammar to be in Chomsky normal form (a restricted form), which is possible but inconvenient for modeling RNA (and also requires more nonterminals). Furthermore, it takes time at least proportional to n^3, whereas the forward-backward procedure for HMMs takes $O(n^2)$ time, where n is the length of the model (and the typical training sequence). There are also a considerable number of local minima, and this presents a problem when the initial grammar is not highly constrained.

In order to avoid such problems, we have developed a different method to obtain a SCFG for an RNA family like tRNA that takes only time n^2, and hence may be practical on larger RNA sequences. Our new algorithm demands folded RNA as training examples, rather than unfolded ones. Thus the base pairs in each training sequence have to be identified before the algorithm can begin iteratively reestimating the grammar parameters. If such base pair information is not available, we can use a fancier version of the algorithm, as described in Section 2.5.

This new algorithm we have developed is based on the theory of stochastic tree-grammars. As the name suggests, tree-grammars are used to derive labeled trees instead of strings. Labeled trees can be used to represent the secondary structure of RNA quite easily [SZ90] (see Figure 2.2). When working with a tree-grammar for RNA, one is explicitly working with the secondary structure of the molecule. Since this structure is given explicitly in each training molecule, we no longer have to (implicitly) sum over *all* possible interpretations of the secondary structure of the training examples when we reestimate the grammar parameters, as must be done with the inside-outside method. The new algorithm also tends to converge faster because each training example is much more informative.

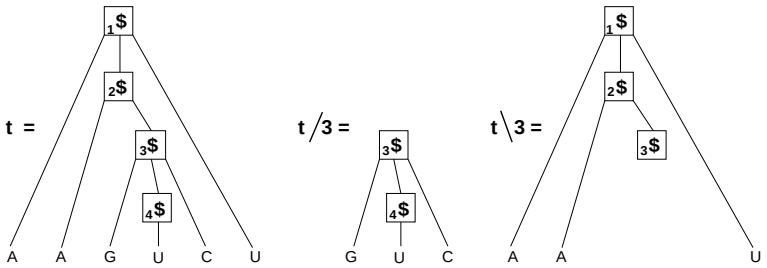


Figure 2.3: The folded RNA sequence `(AA(GUC)U)` can be represented as a tree t (left), which can be broken into two parts such as $t/3$ (middle) and $t\backslash 3$ (right).

To avoid unnecessary complexity, we describe this new algorithm in terms of CFGs instead of tree-grammars [TW68, Sak92]. A tree is a rooted, directed, connected acyclic finite graph in which the direct successors of any node are linearly ordered from left to right. The predecessor of a node

is called the *parent*, the successor, a *child*, and a child of the parent, a *sibling*. A folded RNA sequence can be represented by a labeled tree t as follows. Each leaf node is labeled by one of four nucleotides $\{\mathtt{A}, \mathtt{U}, \mathtt{G}, \mathtt{C}\}$ and all internal nodes are labeled by one special symbol, say \$. The sequence of nucleotides labeled at leaf nodes traced from left to right exactly constitutes the RNA sequence, and the structure of the tree represents its folding structure. See Figure 2.3 for an example of a tree representation of the folded RNA sequence `(AA(GUC)U)`. We assume all internal nodes in t are numbered from 1 to T (the number of internal nodes) in some order, and for an internal node n ($1 \leq n \leq T$), let t/n denote the subtree of t with root n (as shown in the center of Figure 2.3) and let $t \backslash n$ denote the tree obtained by removing a subtree t/n from t (as shown in the right of Figure 2.3).

The probability of any folded sequence t given by a SCFG $G = (N, \Sigma, P, S_0)$ is efficiently calculated using a dynamic programming technique, as is done with the forward algorithm in HMMs. A labeled tree t representing a folded RNA sequence has the shape of a parse tree, so to parse the folded RNA, the grammar G needs only to assign nonterminals to each internal node according to the productions. Let the quantity $in_n(S)$ define the probability of the subtree t/n given that the nonterminal S is assigned to node n and given G, for all nonterminals S and all nodes n such that $1 \leq n \leq T$. We can calculate $in_n(S)$ inductively as follows:

1. Initialization: $in_n(\mathtt{X}) = 1$, for all leaf nodes n and all terminals $\mathtt{X}$ (each nucleotide).
   This extension of $in_n(S)$ is for the convenience of the inductive calculation of $in_n(S)$.
2. Induction:

$$in_m(S) = \sum_{Y_1,\ldots,Y_k \in (N \cup \Sigma)} in_{n_1}(Y_1) \cdots in_{n_k}(Y_k) \cdot \mathcal{P}(S \rightarrow Y_1 \cdots Y_k),$$

   for all nonterminals S, all internal nodes m, and all m's children nodes $n_1, \ldots, n_k$.
3. Termination: for the root node n and the start symbol S_0,

$$\mathrm{Prob}(t \mid G) = in_n(S_0). \tag{2.1}$$

This effective calculation enables us to estimate the new parameters of a SCFG in time proportional to the square of the number of nonterminals in the grammar multiplied by the total size of all the folded training sequences. We need one more quantity, $out_n(S)$, which defines the probability of $t \backslash n$ given that the nonterminal S is assigned to node n and given G:

1. Initialization: for the root node n,

$$out_n(S) = \begin{cases} 1 & \text{for } S = S_0 \text{ (the start symbol)}, \\ 0 & \text{otherwise.} \end{cases}$$

2. Induction:

$$out_m(S) = \sum_{Y_1,\ldots,Y_k \in (N \cup \Sigma), S' \in N} in_{n_1}(Y_1) \cdots in_{n_k}(Y_k) \cdot \mathcal{P}(S' \rightarrow Y_1 \cdots S \cdots Y_k) \cdot out_l(S'),$$

   for all nonterminals S, all internal nodes l and m such that l is the parent of m, and all nodes $n_1, \ldots, n_k$ such that $n_1, \ldots, n_k$ are the siblings of m.

Given a set of folded training sequences $t(1), \ldots, t(n)$, we can see how well a grammar fits them by calculating the probability that it generates them. This probability is simply a product of terms of the form given by (2.1), i.e.,

$$\text{Prob(sequences} \mid G) = \prod_{j=1}^{n} \text{Prob}(t(j) \mid G), \tag{2.2}$$

where each term Prob($t(j)$ | G) is calculated as in Equation (2.1). The goal is to obtain a high value for this quantity, called the *likelihood* of the grammar. The *maximum likelihood* (ML) method of model estimation finds the model that maximizes the likelihood (2.2). There is no known way to directly and efficiently calculate the best model, i.e., the one that maximizes the likelihood. However, the general EM method, given an arbitrary starting point, finds a local maximum by iteratively re-estimating the model such that the likelihood increases in each iteration. This method is often used in statistics. Here we give a version of the EM method to estimate the parameters of a SCFG from folded training RNA sequences. It proceeds as follows:

1. An initial grammar is created by assigning values to the production probability $\mathcal{P}(S \rightarrow Y_1 \ \cdots \ Y_k)$ for all S and all $Y_1, \ \ldots \ , Y_k$, where S is a nonterminal and Y_i $(1 \leq i \leq k)$ is a nonterminal or terminal. If some constraints or features present in the folded sequences are known, these are encoded in the initial grammar. The current grammar is set to this initial grammar.
2. Using the current grammar, the values $in_n(S)$ and $out_n(S)$ for each nonterminal S and each node n for each folded training sequence are calculated in order to get a new estimate of the production probability, $\hat{\mathcal{P}}(S \rightarrow Y_1 \ \cdots \ Y_k)$=

$$\frac{\sum_{\text{sequences } t} \left(\sum_{\text{nodes } n} out_m(S) \cdot \mathcal{P}(S \rightarrow Y_1 \ \cdots \ Y_k) \ \cdot \ in_{n_1}(Y_1) \ \cdots \ in_{n_k}(Y_k) \ / \ \text{Prob}(t \mid G) \right)}{\text{norm}},$$

   where G is the old grammar and "norm" is the appropriate normalizing constant so that $\sum_{Y_1,\ldots,Y_k} \hat{\mathcal{P}}(S \rightarrow Y_1 \ \cdots \ Y_k) = 1$.
3. A new current grammar is created by simply replacing $\mathcal{P}(S \rightarrow Y_1 \cdots Y_k)$ with the re-estimated probability $\hat{\mathcal{P}}(S \rightarrow Y_1 \cdots Y_k)$.
4. Steps 2 and 3 are repeated until the parameters of the current grammar change only insignificantly.

2.4 Overfitting and regularization

A grammar with too many free parameters cannot be estimated well from a relatively small set of training sequences. Attempts to estimate such a grammar will encounter the problem of *overfitting*, in which the grammar fits the training sequences very well, but gives a poor fit to related (test) sequences that were not included in the training set. One solution is to control the effective number of free parameters by *regularization*. We calculated a regularizer from the multiple alignment of tRNA sequences and added it to the counts used for reestimating the probabilities of productions of the grammar in each iteration of training. Similar methods are described in [KBM+92].

2.5 Iterative usage of the training algorithm

Since our EM training algorithm uses folded RNA as training examples, rather than unfolded ones, the base pairs in each training sequence need to be identified before the EM iteration begins. If only unfolded training sequences are available, then we iteratively estimate the folding of the training sequences as well using the following method:

1. First, we design a rough initial grammar which might represent only a portion of the base pairing interactions. This is used to parse the initial unfolded RNA training sequences to obtain a set of partially folded RNA sequences.
2. Next, we estimate a SCFG using the partially folded sequences and our training algorithm to obtain a new estimated grammar. Further productions might be added to the grammar at this stage, although we have not experimented with this possibility yet.
3. Then we use the trained grammar to obtain more accurately folded training sequences and estimate a SCFG using these.
4. We repeat this process until the trained grammar gives no changes to the folding.

2.6 Dealing with introns

Introns are sometimes present in tRNA genes. This means that when we search databank files of genomic sequences, the sequence of the tRNA may be interrupted by non-tRNA coding nucleotides. The grammar needs to be extended to handle this situation. Introns are normally present in the anticodon stem loop (reviewed in [PG93]). We make the assumption that an intron (of whatever type) will be present in the anticodon loop and more specifically within or on either side of the anticodon itself, i.e., we consider a total of five possible positions for introns.

An extremely useful advantage of SCFGs is their modularity. We see this clearly in this case: an intron grammar can be deduced separately from the grammar for plain tRNA, then these two separate grammars can be combined into a single grammar simply by uniting the two sets of independent productions and maintaining their different probability distributions.

3 Results

As described in the previous section, we derived two grammars for tRNA sequences: the *alignment grammar*, directly derived from an existing multiple alignment of tRNAs, and the *trained grammar*, deduced by our algorithm from a training set of unfolded and unaligned tRNA sequences. We compared the two grammars by evaluating their abilities to perform three tasks: to discriminate tRNA sequences from non-tRNA sequences, to produce multiple alignments, and to ascertain the secondary structure of new tRNAs.

To derive the trained grammar, we designed the initial grammar by using some prior knowledge about the tRNA family: tRNA has four principle arms and one extra arm, and only the lengths of the D arm and the Extra arm vary. The initial grammar was set up with a total of 75 nonterminals and 660 productions to derive arms of the appropriate lengths. Productions of the Type I shown Figure 3.1 are used to derive those five arms, productions of the Type II are used describe the base-pairing in four principle arms, and productions of Type III and IV are used to describe a loop of unpaired bases. A uniform probability distribution was placed over each set of productions

Type I		Type II				Type III		Type IV	
$S \rightarrow SS$	1.0	$S \rightarrow$ A S A	0.05	$S \rightarrow$ C S A	0.05	$S \rightarrow$ A S	0.2*	$S \rightarrow$ A	0.25
		$S \rightarrow$ A S G	0.05	$S \rightarrow$ C S G	0.1	$S \rightarrow$ G S	0.2*	$S \rightarrow$ G	0.25
		$S \rightarrow$ A S C	0.05	$S \rightarrow$ C S C	0.05	$S \rightarrow$ C S	0.2*	$S \rightarrow$ C	0.25
		$S \rightarrow$ A S U	0.1	$S \rightarrow$ C S U	0.05	$S \rightarrow$ U S	0.2*	$S \rightarrow$ U	0.25
		$S \rightarrow$ G S A	0.05	$S \rightarrow$ U S A	0.1	$S \rightarrow S$	0.2*		
		$S \rightarrow$ G S G	0.05	$S \rightarrow$ U S G	0.05				
		$S \rightarrow$ G S C	0.1	$S \rightarrow$ U S C	0.05				
		$S \rightarrow$ G S U	0.05	$S \rightarrow$ U S U	0.05				

Figure 3.1: To obtain production probabilities for this initial grammar, we placed a uniform distribution over each subset of same-type productions, but weighted Watson-Crick base pairs twice as heavily. Some values (*) may differ if no skip productions are needed. In this case, the $S \rightarrow S$ production probability is zero, but the distribution remains uniform on the remaining productions. For simplicity, nonterminal subscripts were omitted in this figure.

of the same type, except Watson-Crick base pairs, which were assigned higher probabilities (see Figure 3.1). This initial grammar alone found the correct folding for 93% of our whole database of 1477 tRNA sequences.[2] The EM method described in the previous section was then used to refine this grammar using 500 training tRNA sequences. The run-time was around 10 CPU minutes on a Sun Sparcstation. During this process, only the probabilities of the productions were reestimated and no nonterminals or productions were added or deleted (unlike "model surgery" in [KBM+92]). Our future work will focus on developing some method that can automatically select a good structure and a good length of the grammar while training.

3.1 Data

The experiments used data from three sources:

1. From EMBL Data Library's database produced by Mathias Sprinzl and co-workers, Bayreuth, FRG, we obtained 1477 aligned and folded complete tRNA sequences. A *complete* tRNA sequence means a tRNA sequence which has the four major arms and a length between 71 and 95 bases.[3] Thus we included tRNAs of virus, bacteriophage, archaebacteria, eubacteria, cyanelle, chloroplast, cytoplasm, and some mitochondria, and did not include other mitochondrial tRNAs. We changed several specific symbols used for representing modified bases to the usual `A`, `C`, `G`, `U` symbols. Of these 1477 tRNA sequence descriptions, we selected randomly 500 as training examples for deriving a grammar to model intron-free tRNA and used the rest as test data.
2. The Ribosomal Database Project's (RDP) [OOL+92] aligned, folded large subunit ribosomal RNA data file `LSU.aln` provided primary source of non-tRNA sequences. We generated

[2]The initial grammar was not able to find the correct folding for the rest (7%) of the tRNA sequences, and failed to discriminate tRNAs from non-tRNAs.

[3]Some tRNAs are lacking some arms (e.g., mitochondrial tRNA may sometimes have only three loops). SCFGs can model such irregular tRNAs, too. However in this paper, we do not deal with this problem.

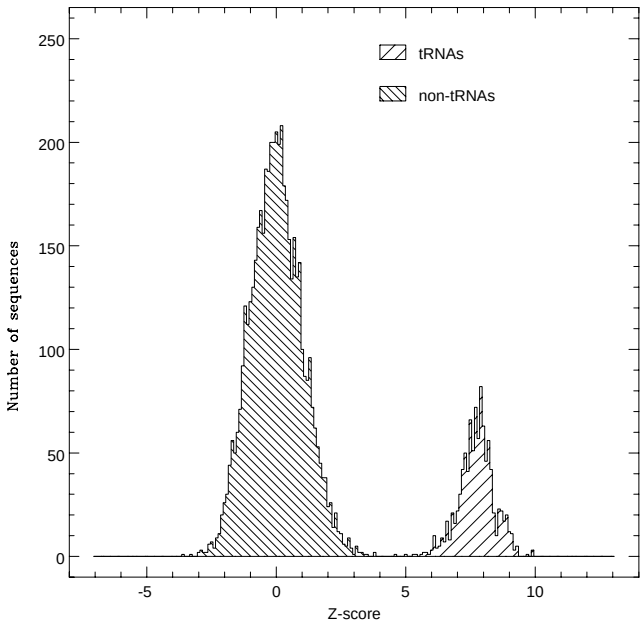
approximately 2,400 non-tRNA sequences by cutting ribosomal RNA sequences into pieces of approximately the same lengths as tRNA sequences.

3. From the National Center for Biotechnology Information's (NCBI) NewGenBank and GenBank databases, we used 55 unaligned and unfolded tRNA sequences with introns of rather short lengths (from 4 to 25). The GenBank databases also include descriptions of other RNA besides tRNA. From these descriptions, we generated an additional 2,500 non-tRNA sequences.

3.2 Discriminations of tRNAs from non-tRNAs: Database search

As described in Section 2.2, we calculate a NLL-score for each test sequence and use it to measure how well the sequence fits the grammar. This raw NLL-score depends too much on the length of test sequence to be used directly to decide whether a sequence belongs to the family modeled by the grammar. However, this problem can be overcome by normalizing the NLL-score appropriately. Details are described in [KBM+92]. Essentially, we calculate the difference between the NLL-score of a sequence and the average NLL-score of a typical non-tRNA sequence of the same length, measured in standard deviations. This number is called the *Z-score* for the sequence. We then choose a Z-score cutoff, and sequences with Z-scores above the cutoff are classified as positive examples.

Figure 3.2: The number of sequences with a certain Z-score scored by the alignment grammar. The test set of 977 tRNA sequences cluster around a Z-score near 7.5, while 4885 non-tRNA sequences cluster around a Z-score near 0.

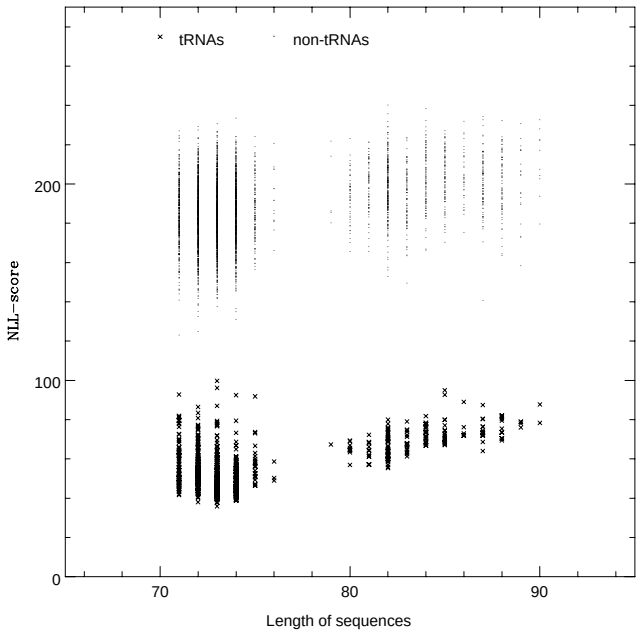


For the alignment grammar and the trained grammar, NLL-scores and Z-scores were computed for 977 test tRNA sequences and 4885 non-tRNA sequences of length 71 to 90. For each tRNA sequence, there are five non-tRNA sequences of the same length. Figure 3.2 shows the Z-score histogram for the alignment grammar. The grammar distinguishes perfectly between tRNAs and non-tRNAs: the lowest Z-score of tRNAs is 4.984 and the highest Z-score of non-tRNAs is 4.589.

Thus, choosing a Z-score cutoff between them, we can discriminate tRNA sequences from non-tRNA sequences perfectly.

Surprisingly, the trained grammar was able to discriminate more reliably than the alignment grammar in that the trained grammar created a greater gap between Z-scores of tRNAs and non-tRNAs. NLL-scores and Z-scores made by the trained grammar are shown in Figures 3.3 and 3.4. Since the range of lengths of tRNA sequences is short, NLL-scores would be sufficient to distinguish tRNAs from non-tRNAs (while NLL-scores made by the alignment grammar did not distinguish well). However, the Z-scores are of independent interest for statistical reasons. The lowest Z-score of tRNAs is 5.464 and the highest Z-score of non-tRNAs is 4.517. Thus the trained grammar distinguishes perfectly between tRNAs and non-tRNAs and more reliably than the alignment grammar. This is unexpected because the trained grammar is obtained using only 500 training sequences, so 977 test tRNA sequences are completely new for the trained grammar, while the alignment grammar is obtained using all 1477 aligned tRNA sequences.

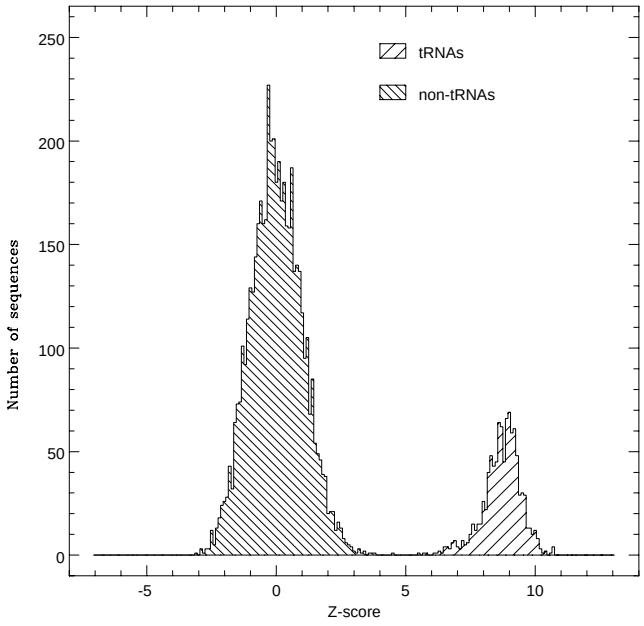
Figure 3.3: NLL-score versus sequence length for tRNAs and non-tRNAs. All complete tRNA sequences of length 71-90 from the EMBL Data Library are shown.



3.3 Multiple sequence alignments

From a grammar it is possible to obtain a multiple alignment of all the sequences. The grammar can produce the most likely parse tree for the sequences to be aligned. This gives an alignment of all the nucleotides that align to the match nonterminals on the main line in the grammar. Between the match nonterminals there might be insertions of varying lengths, but by inserting enough spaces in all the sequences to accommodate the longest insertion, an alignment is obtained. Figure 3.5 shows the original alignment of 15 tRNA sequences in EMBL Data Library. Figure 3.6 shows the alignment produced by the trained grammar for the same sequences. The boundaries of the helices and loops are the same as those in Figure 3.5. The major difference between the two alignments is the extra arm, which is itself highly variable in terms of its length and sequence. Both alignment

Figure 3.4: The number of sequences with a certain Z-score scored by the trained grammar. Shown is the test set of 977 tRNA sequences and 4885 non-tRNA sequences.




```
Organism          |         <     D arm       > < Anticodon arm ><     Extra arm        ><   T arm     >
                  |     Base pairings                 Anticodon                          Base pairings
                    (((((((  ((((           )))) (((((  ===  )))))                    (((((       ))))))))))))
ACHOLEPLASMA LAID.  GCCCGGGUGGUGAAAUC-GGUA-GACACGCAGGACUUAAAAUCCUGU-GGCA---UAAA----AGCCA-UGUCGGUUCAAGUCCGACCCCGGGCA
ARABIDOPSIS THAL.   GGAUGGAUGUCUGAGC--GGUUGAAAGAGUCGGUCUUGAAAACCGAA-GUAUUU-CUAG--GAAUAC--CGGGGGUUCGAAUCCCUCUCCAUCCG
BACILLUS SUBTILIS   GCGGAUGUGGCGGAAUU-GGCA-GACGCGCUAGAAUCAGGCUCUAGU-GUCU---UUAC----AGACG-UGGGGGUUCAAGUCCCUUCAUCCGCA
BOMBYX MORI         GCCCGGCUAGCUCAGUC-GGU--AGAGCAUGAGACUCUUAAUCUCAGG------------------GUCGUGGGUUCGAGCCCCACGUUGGGCG
BRASSICA NAPUS      GGGUGUAUAGCUCAGUU-GGU--AGAGCAUUGGGCUUUUAACCUAAUG------------------GUCGCAGGUUCAAGUCCUGCUAUACCCA
DICTYOSTELIUM DIS.  GACAUCGUAGCAAAGU--GGUCUAAUGCGUCUGACUAGAAAUCAGAU-CCCU---UC------GGGGG-CGCAGGUUCGAACCCUGCCGAUGUCG
E.COLI              GCCGAGGUGGUGGAAUU-GGUA-GACACGCUACCUUGAGGUGGUAGU-GCCC---AAUA----GGGCU-UACGGGUUCAAGUCCCGUCCUCGGUA
E.COLI              GCGAAGGUGGCGGAAUU-GGUA-GACGCGCUAGCUUCAGGUGUUAGU-GUCC---UUAC----GGACG-UGGGGGUUCAAGUCCCCCCCCUCGCA
E.COLI              GGAGAGAUGCCGGAGC--GGCUGAACGGACCGGUCUCGAAAACCGGA-GUAGGG-GCAA--CUCUAC--CGGGGGUUCAAAUCCCCCUCUCUCCG
HAEMOPHILUS INFLU.  GGGUCGUUAGCUCAGAC-GGU--AGAGCAGCGGACUUUUAAUCCGUUG------------------GUCGAAGGUUCGAAUCCUUCACGACCCA
HALOBACTERIUM VOL.  GCGGGGGUGGCUGAGCCAGGCCAAAAGCGGCGGACUUAAGAUCCGCU-CCC----GUAG-----GGGUUCGCGAGUUCGAAUCUCGUCCCCCGCA
HALOBACTERIUM VOL.  GCGCGGGUAGCCAAGU--GGCCAAAGGCGCAGCGCUUAGGACGCUGU-GGU----GUAG-----ACCUUCGCAGGUUCGAACCCUGUCCCGCGCA
METHANOCOC.VANI.    GGGCCCGUAGCUUAGUCUGGU--AGAGCGCCUGGCUUUUAAUCAGGCG------------------GUCGAGGGUUCGAAUCCCUUCGGGCCCG
MYCOPLASMA MYCOID.  GGAAGAUUACCCAAGUCCGGCUGAAGGGAUCGGUCUUGAAAACCGAGAGUCGG--GGAAA--CCGAG--CGGGGGUUCGAAUCCCUCAUCUUCCG
RHODOSPIRIL. RUB.   GCCUUUGUAGCGGAAU--GGU--AACGCGGCAGACUCAAAAUCUGCU-UUGG---UAAC----CCAGG-UGGUAGUUCGACUCUCCCCAAAGGCA
```

Figure 3.5: The original alignment of 15 tRNA sequences from EMBL Data Library. The parentheses above the alignments indicate which columns (positions) form base pairs and "`===`" indicates the anticodon domain.

give the same base-pairing. Once a grammar has been constructed, a similar multiple alignment can be produced for the entire set of 1477 tRNA sequences (or any subset).

3.4 Predictions of secondary structures

As discussed in the last section, the trained grammar produces the same alignment as the original alignment for the base pairing parts. This implies that the most likely parse trees produced by the

```
     Base pairings                    Anticodon                                                  Base pairings
(((((((  ((((            )))) (((((  ===  )))))                                                (((((       ))))))))))))
GCCCGGGUGGUGAAAU.C.GG.Ua.GACACGCAGGACUUAAAAUCCUGU..........Ggcauaaaagcc..A-............-UGUCGGUUCAAGUCCGACCCCGGGCA
GGAUGGAUGUCUGAGC.-.GGuUg.AAAGAGUCGGUCUUGAAAACCGAA..........Guauuucuaggaau A-............CCGGGGGUUCGAAUCCCUCUCCAUCCG
GCGGAUGUGGCGGAAU.U.GG.Ca.GACGCGCUAGAAUCAGGCUCUAGU..........Gucuuuacagac..G-............-UGGGGGUUCAAGUCCCUUCAUCCGCA
GCCCGGCUAGCUCAGU.C.GG.U..AGAGCAUGAGACUCUUAAUCUCAG..........G............G-............UCGUGGGUUCGAGCCCCACGUUGGGCG
GGGUGUAUAGCUCAGU.U.GG.U..AGAGCAUUGGGCUUUUAACCUAAU..........G............G-............UCGCAGGUUCAAGUCCUGCUAUACCCA
GACAUCGUAGCAAAGU.-.GG.UcuAAUGCGUCUGACUAGAAAUCAGAUcccuucggg..G............G-............-CGCAGGUUCGAACCCUGCCGAUGUCG
GCCGAGGUGGUGGAAU.U.GG.Ua.GACACGCUACCUUGAGGUGGUAGUgcccaauag..G............GC............UUACGGGUUCAAGUCCCGUCCUCGGUA
GCGAAGGUGGCGGAAU.U.GG.Ua.GACGCGCUAGCUUCAGGUGUUAGU..........Guccuuacggac..G-............-UGGGGGUUCAAGUCCCCCCCCUCGCA
GGAGAGAUGCCGGAGC.-.GGcUg.AACGGACCGGUCUCGAAAACCGGA..........Guaggggcaacucu A-............CCGGGGGUUCAAAUCCCCCUCUCUCCG
GGGUCGUUAGCUCAGA.C.GG.U..AGAGCAGCGGACUUUUAAUCCGUU..........G............G-............UCGAAGGUUCGAAUCCUUCACGACCCA
GCGGGGGUGGCUGAGCcA.GG.CcaAAAGCGGCGGACUUAAGAUCCGCUcccguaggg..G............U-............UCGCGAGUUCGAAUCUCGUCCCCCGCA
GCGCGGGUAGCCAAGU.-.GG.CcaAAGGCGCAGCGCUUAGGACGCUGU..........G............GUguagaccu.....UCGCAGGUUCGAACCCUGUCCCGCGCA
GGGCCCGUAGCUUAGUcU.GG.U..AGAGCGCCUGGCUUUUAAUCAGGC..........G............G-............UCGAGGGUUCGAAUCCCUUCGGGCCCG
GGAAGAUUACCCAAGU.CcGGcUg.AAGGGAUCGGUCUUGAAAACCGAG..........A............GUcggggaaaccgag-CGGGGGUUCGAAUCCCUCAUCUUCCG
GCCUUUGUAGCGGAAU.-.GG.U..AACGCGGCAGACUCAAAAUCUGCUuugguaacccaG............G-............-UGGUAGUUCGACUCUCCCCAAAGGCA
```

Figure 3.6: Alignment produced by the trained grammar for the tRNA sequences depicted in Figure 3.5. Capital letters correspond to nucleotides aligned to the main line of the grammar, "-" to deletions by skip productions in the grammar, and lower-case letters to nucleotides treated as insertions by the grammar. The "." is used as a fill character to accommodate insertions.

trained grammar give the correct secondary structure for all 1477 tRNA sequences, 500 training plus 977 unseen test sequences.

3.5 Introns

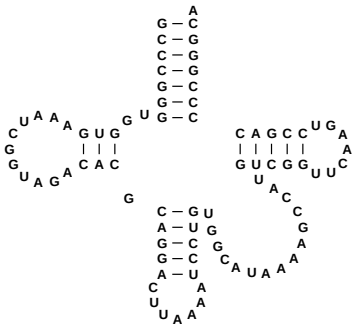
Our experiments with introns are currently only very preliminary. We have not yet obtained a large enough data set to do truly meaningful experiments. In one experiment, we used 55 sequences of introns and a simple regular grammar with five nonterminals for training a (sub)grammar to model introns. The training process reestimated the distributions of the four nucleotides at the first and last two consective positions of the introns. This grammar trained for introns and the grammar previously trained for intron-free tRNAs were then combined into a single grammar modeling tRNAs with introns.

We tested the grammar on the same 55 tRNA sequences with introns. Of these, the grammar correctly identified the positions of introns and introns themselves for 44 sequences. Introns with the same lengths as the correct ones but incorrect positions for two or three bases (shifted) were found for 7 sequences. The grammar completely failed to identify introns for 4 sequences. Thus, this approach shows some promise in identifying introns and finding the correct secondary structure for tRNA sequences with introns.

3.6 Displaying folded RNA sequences: XRNA

XRNA is a suite of programs providing, among other functions, graphic manipulation and labeling of previously determined RNA secondary structures [WGN93]. Using simple filters, we were able to transform the secondary structure predicted by our grammar into XRNA format. Figure 3.7 shows XRNA's depiction of a secondary structure predicted by the trained grammar.

Figure 3.7: After training on 500 examples picked at random from the EMBL database (no introns), the trained grammar predicted this folding for a previously unseen tRNA sequence, the first sequence listed in Figure 3.6. XRNA generated this figure.



4 Discussion

The method we have proposed represents a significant new direction in computational biosequence analysis. We believe SCFGs may provide a flexible and highly effective statistical method in a number of problems for RNA sequences including database searching, multiple alignment, prediction of secondary structures, and dealing with introns, and that the grammar itself may be a valuable tool for representing an RNA family or domain. The present work demonstrates the usefulness of SCFGs with tRNA sequences. Since our experiments with introns are only preliminary, further work will be required to demonstrate the usefulness of SCFGs in searching databases for tRNA sequences with introns. Extending the grammar to handle the more unusual mitochondrial tRNAs would also be of interest. However, the most challenging future problem is to model a family of larger RNA sequences, e.g. ribosomal RNA, with SCFGs.

Another important area for future research will be to determine how much prior knowledge about the structure of the class of RNA sequences being modeled is necessary for this approach to work. In our experiments with tRNA, we started the training with an initial grammar that contained quite a bit of knowledge about the structure of tRNA. A more challenging training approach would be to use a homogeneous initial grammar embodying no specific knowledge about the tRNA family. As described in Section 2.5, we could then try to gradually extend the grammar to account for the structure of the training sequences. We might do this by starting with a regular grammar that represents an HMM like those used to model protein families in our previous work [KBM+92]. Then, by studying multiple alignments produced by this grammar, we might be able to use methods for finding correlations between columns in this multiple alignment, such as those in [GPH+92, Lap92, KB93], to discover some of the base paring structure in tRNA. Having done this, it would be straightforward to modify the grammar to account for this base pairing, and then iterate this process until no new structure is found. This process would essentially automate some of the hand methods used in the original investigation of tRNA structure by phylogenetic analysis.

Finally, there is the question of what further generalizations of hidden Markov models, beyond SCFGs, might be useful in computational biosequence analysis. The key advantage of the method we propose here over the HMM method is that it allows us to explicitly deal with the secondary structure of the RNA sequence. We make the leap from stochastic models of strings to stochastic models of trees, and this lets us model the base pairing interactions of the molecule, which determine

its secondary structure. This progression is similar to the path taken by the late King Sun Fu and his colleagues in their development of the field of syntactic pattern recognition [Fu82]. To go beyond this to the tertiary structure would require still more general methods. One possibility would be to consider *stochastic graph grammars* (see e.g. [ER91]) in hopes of obtaining a more general model of the interactions present in the molecule beyond the primary structure. If a stochastic graph grammar framework could be developed that included both an efficient method of finding the most probable "folding" of the molecule given the grammar and an efficient EM method for estimating the parameters of the grammar from folded examples, then extensions of the approach taken in this paper to more challenging problems, including RNA tertiary structure and protein folding, would be possible. This is perhaps the most interesting direction for further research suggested by the results of this paper.

Acknowledgments

We thank Anders Krogh, Richard Hughey, Harry Noller, Kimmen Sjölander for helpful discussion and Bryn Weiser for help with the XRNA system.

References

[AU72] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation and Compiling, Vol. I: Parsing*. Prentice Hall, Englewood Cliffs, N.J., 1972.

[Bak79] J. K. Baker. Trainable grammars for speech recognition. *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550, 1979.

[BCHM93] P. Baldi, Y. Chauvin, T. Hunkapiller, and M. A. McClure. Hidden Markov models in molecular biology: new algorithms and applications. In Hanson, Cowan, and Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 747–754, San Mateo, CA, 1993. Morgan Kauffmann Publishers.

[BHK+93] M. P. Brown, R. Hughey, A. Krogh, I. S. Mian, K. Sjölander, and D. Haussler. Using Dirichlet mixture priors to derive hidden Markov models for protein families. Proc. of workshop on AI in Molecular Biology, Wash. D.C., July 1993. to appear.

[CS92] L. R. Cardon and G. D. Stormo. Expectation maximization algorithm for identifying protein-binding sites with variable lengths from unaligned DNA fragments. *Journ Mol Biol*, 223:159–170, 1992.

[ER91] J. Engelfriet and G. Rozenberg. Graph grammars based on node rewriting: and introduction to NLC graph grammars. In E. Ehrig, H.J. Kreowski, and G. Rozenberg, editors, *Lecture Notes in Computer Science*, volume 532, pages 12–23. Springer-Verlag, 1991.

[Fu82] K.S. Fu. *Syntactic pattern recognition and applications*. Prentice-Hall, 1982.

[GPH+92] R. R. Gutell, A. Power, G. Z. Hertz, E. J. Putz, and G. D. Stormo. Identifying constraints on the higher-order structure of RNA: continued development and application of comparative sequence analysis methods. *Nucleic Acids Research*, 20:5785–5795, 1992.

[HKMS93] D. Haussler, A. Krogh, I. S. Mian, and K. Sjölander. Protein modeling using hidden Markov models: Analysis of globins. In *Proceedings of the Hawaii International Conference on System Sciences*, Los Alamitos, CA, 1993. IEEE Computer Society Press.

[JOP89] B. D. James, G. J. Olsen, and N. R. Pace. Phylogenetic comparative analysis of RNA secondary structure. *Methods in Enzymology*, 180:227–239, 1989.

[JTZ90] J. A. Jaeger, D. H. Turner, and M. Zuker. Predicting optimal and suboptimal secondary structure for RNA. *Methods in Enzymology*, 183:281–306, 1990.

[KB93] Tod Klinger and Douglas Brutlag. Unpublished manuscript, 1993.

[KBM+92] A. Krogh, M. Brown, I. S. Mian, K. Sjolander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. Submitted to J. Mol. Bio., December 1992.

[Lap92] Allan Lapedes. Private communication, 1992.

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

[OOL+92] G. J. Olsen, R. Overbeek, N. Larsen, T. L. Marshand M. J. McCaughey, M. A. Maciukenas, W. M. Kuan, T. J. Macke, Y. Xing, and C. R. Woese. The ribosomal database project. *Nucleic Acids Research*, 20:2199–200, 1992.

[PG93] E. M. Phizicky and C. L. Greer. Pre-tRNA splicing: variation on a theme or exception to the rule? *Trends in Biochemical Sciences*, 18:31–34, 1993.

[Rab89] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc IEEE*, 77(2):257–286, 1989.

[Sak92] Yasubumi Sakakibara. Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, 97:23–60, 1992.

[San85] David Sankoff. Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM J. Appl. Math.*, 45:810–825, 1985.

[Sea92] David B. Searls. The linguistics of DNA. *American Scientist*, 80:579–591, November–December 1992.

[SZ90] Bruce A. Shapiro and Kaizhong Zhang. Comparing multiple RNA secondary structures using tree comparisons. *CABIOS*, 6(4):309–318, 1990.

[TW68] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2:57–81, 1968.

[Wat89] M. S. Waterman. Sequence alignments. In M S Waterman, editor, *Mathematical Methods for DNA Sequences*. CRC Press, 1989.

[WGGN83] C. R. Woese, R. R. Gutell, R. Gupta, and H. F. Noller. Detailed analysis of the higher-order structure of 16s-like ribosomal ribonucleic acids. *Microbiology Reviews*, 47(4):621–669, 1983.

[WGN93] Bryn Weiser, Robin Gutell, and Harry Noller. XRNA: an X-windows environment RNA editing/display package. Unpublished manuscript, January 1993.

[ZS84] Michael Zuker and David Sankoff. RNA secondary structures and their prediction. *Bull. Math. Biol.*, 46:591–621, 1984.

A Appendix

The trained grammar was deduced from 500 tRNA sequences without introns. Tabulated here are its total 660 productions. The start symbol S_0 is written as S and the other nonterminals are represented in abbreviated form—namely, as the numbers that in full form would appear as S-symbol subscripts. Nonterminals numbered from 1 to 22, from 26 to 46 and from 63 to 79 are match nonterminals and nonterminals numbered from 81 to 87 and from 90 to 96 are insert nonterminals.

S → 1 75	1.000000	8 → 9 76	1.000000	76 → 11 77	1.000000	77 → 26 78	1.000000
78 → 27 79	1.000000	79 → 39 63	1.000000	1 → C 2 C	0.004207	1 → C 2 G	0.066512
1 → C 2 U	0.004282	1 → C 2 A	0.016078	1 → G 2 C	0.650489	1 → G 2 G	0.000308
1 → G 2 U	0.059865	1 → G 2 A	0.000282	1 → U 2 C	0.000360	1 → U 2 G	0.000947
1 → U 2 U	0.004554	1 → U 2 A	0.075745	1 → A 2 C	0.004450	1 → A 2 G	0.000287
1 → A 2 U	0.111253	1 → A 2 A	0.000380	2 → C 3 C	0.000303	2 → C 3 G	0.380810
2 → C 3 U	0.000378	2 → C 3 A	0.002413	2 → G 3 C	0.412326	2 → G 3 G	0.000308
2 → G 3 U	0.028631	2 → G 3 A	0.000282	2 → U 3 C	0.000360	2 → U 3 G	0.012660
2 → U 3 U	0.000650	2 → U 3 A	0.087458	2 → A 3 C	0.000545	2 → A 3 G	0.000287
2 → A 3 U	0.072210	2 → A 3 A	0.000380	3 → C 4 C	0.000303	3 → C 4 G	0.332006
3 → C 4 U	0.000378	3 → C 4 A	0.004365	3 → G 4 C	0.332287	3 → G 4 G	0.000308
3 → G 4 U	0.061818	3 → G 4 A	0.000282	3 → U 4 C	0.000360	3 → U 4 G	0.010708
3 → U 4 U	0.002602	3 → U 4 A	0.130405	3 → A 4 C	0.000545	3 → A 4 G	0.000287
3 → A 4 U	0.121014	3 → A 4 A	0.002333	4 → C 5 C	0.002255	4 → C 5 G	0.242207
4 → C 5 U	0.002330	4 → C 5 A	0.004365	4 → G 5 C	0.330335	4 → G 5 G	0.000308
4 → G 5 U	0.048152	4 → G 5 A	0.000282	4 → U 5 C	0.000360	4 → U 5 G	0.061464
4 → U 5 U	0.000650	4 → U 5 A	0.169448	4 → A 5 C	0.002497	4 → A 5 G	0.000287
4 → A 5 U	0.134679	4 → A 5 A	0.000380	5 → C 6 C	0.006159	5 → C 6 G	0.294916
5 → C 6 U	0.002330	5 → C 6 A	0.000461	5 → G 6 C	0.254200	5 → G 6 G	0.000308
5 → G 6 U	0.052057	5 → G 6 A	0.004187	5 → U 6 C	0.000360	5 → U 6 G	0.024373
5 → U 6 U	0.002602	5 → U 6 A	0.128453	5 → A 6 C	0.002497	5 → A 6 G	0.000287
5 → A 6 U	0.226430	5 → A 6 A	0.000380	6 → C 7 C	0.000303	6 → C 7 G	0.253920
6 → C 7 U	0.000378	6 → C 7 A	0.000461	6 → G 7 C	0.238582	6 → G 7 G	0.000308
6 → G 7 U	0.032535	6 → G 7 A	0.000282	6 → U 7 C	0.000360	6 → U 7 G	0.053655
6 → U 7 U	0.014315	6 → U 7 A	0.224109	6 → A 7 C	0.000545	6 → A 7 G	0.000287
6 → A 7 U	0.179579	6 → A 7 A	0.000380	7 → C 8 C	0.000303	7 → C 8 G	0.029421
7 → C 8 U	0.000378	7 → C 8 A	0.000461	7 → G 8 C	0.373282	7 → G 8 G	0.000308
7 → G 8 U	0.032535	7 → G 8 A	0.002235	7 → U 8 C	0.000360	7 → U 8 G	0.000947
7 → U 8 U	0.000650	7 → U 8 A	0.206540	7 → A 8 C	0.000545	7 → A 8 G	0.000287
7 → A 8 U	0.351369	7 → A 8 A	0.000380	9 → C 10	0.000671	9 → G 10	0.010686
9 → U 10	0.963879	9 → A 10	0.024763	10 → C	0.038552	10 → G	0.335656
10 → U	0.016880	10 → A	0.608913	11 → C 12 C	0.000303	11 → C 12 G	0.025517
11 → C 12 U	0.000378	11 → C 12 A	0.002413	11 → G 12 C	0.763715	11 → G 12 G	0.000308
11 → G 12 U	0.122335	11 → G 12 A	0.000282	11 → U 12 C	0.000360	11 → U 12 G	0.004851
11 → U 12 U	0.002602	11 → U 12 A	0.023036	11 → A 12 C	0.000545	11 → A 12 G	0.000287
11 → A 12 U	0.052688	11 → A 12 A	0.000380	12 → C 13 C	0.000303	12 → C 13 G	0.650210
12 → C 13 U	0.000378	12 → C 13 A	0.000461	12 → G 13 C	0.035558	12 → G 13 G	0.000308
12 → G 13 U	0.001300	12 → G 13 A	0.000282	12 → U 13 C	0.000360	12 → U 13 G	0.018516
12 → U 13 U	0.000650	12 → U 13 A	0.245583	12 → A 13 C	0.000545	12 → A 13 G	0.000287

12 → A 13 U 0.044879	12 → A 13 A 0.000380	13 → C 14 C 0.000303	13 → C 14 G 0.218781
13 → C 14 U 0.000378	13 → C 14 A 0.006317	13 → G 14 C 0.215157	13 → G 14 G 0.000308
13 → G 14 U 0.003253	13 → G 14 A 0.002235	13 → U 14 C 0.000360	13 → U 14 G 0.008755
13 → U 14 U 0.000650	13 → U 14 A 0.473986	13 → A 14 C 0.002497	13 → A 14 G 0.000287
13 → A 14 U 0.062449	13 → A 14 A 0.004285	14 → C 15 C 0.004207	14 → C 15 G 0.456945
14 → C 15 U 0.002330	14 → C 15 A 0.004365	14 → G 15 C 0.008227	14 → G 15 G 0.002260
14 → G 15 U 0.014966	14 → G 15 A 0.117412	14 → U 15 C 0.000360	14 → U 15 G 0.094651
14 → U 15 U 0.068975	14 → U 15 A 0.097219	14 → A 15 C 0.004450	14 → A 15 G 0.006143
14 → A 15 U 0.015597	14 → A 15 A 0.101893	15 → C 81 0.004600	15 → G 81 0.000727
15 → U 81 0.004766	15 → A 81 0.007717	15 → 16 0.013649	15 → C 16 0.000709
15 → G 16 0.002578	15 → U 16 0.003044	15 → A 16 0.962209	16 → C 82 0.000684
16 → G 82 0.058356	16 → U 82 0.001130	16 → A 82 0.001870	16 → 17 0.003516
16 → C 17 0.003357	16 → G 17 0.670300	16 → U 17 0.001734	16 → A 17 0.259052
17 → C 83 0.035436	17 → G 83 0.000778	17 → U 83 0.039975	17 → A 83 0.002145
17 → 18 0.005830	17 → C 18 0.169033	17 → G 18 0.036450	17 → U 18 0.656792
17 → A 18 0.053561	18 → C 84 0.011019	18 → G 84 0.000716	18 → U 84 0.023407
18 → A 84 0.002634	18 → 19 0.550425	18 → C 19 0.052808	18 → G 19 0.001300
18 → U 19 0.322187	18 → A 19 0.035503	19 → C 85 0.001171	19 → G 85 0.000842
19 → U 85 0.003227	19 → A 85 0.002010	19 → 20 0.001485	19 → C 20 0.000718
19 → G 20 0.964033	19 → U 20 0.001679	19 → A 20 0.024836	20 → C 86 0.003906
20 → G 86 0.226948	20 → U 86 0.000935	20 → A 86 0.001296	20 → 21 0.001464
20 → C 21 0.003233	20 → G 21 0.756208	20 → U 21 0.001440	20 → A 21 0.004570
21 → C 87 0.058272	21 → G 87 0.001425	21 → U 87 0.268719	21 → A 87 0.004610
21 → 22 0.001129	21 → C 22 0.031293	21 → G 22 0.063284	21 → U 22 0.535341
21 → A 22 0.035927	22 → C 0.008646	22 → G 0.060528	22 → U 0.020867
22 → A 0.909958	26 → C 0.032570	26 → G 0.453283	26 → U 0.090646
26 → A 0.423500	27 → C 28 C 0.004207	27 → C 28 G 0.378858	27 → C 28 U 0.008186
27 → C 28 A 0.006317	27 → G 28 C 0.074601	27 → G 28 G 0.000308	27 → G 28 U 0.063770
27 → G 28 A 0.002235	27 → U 28 C 0.000360	27 → U 28 G 0.045847	27 → U 28 U 0.012362
27 → U 28 A 0.284626	27 → A 28 C 0.000545	27 → A 28 G 0.002239	27 → A 28 U 0.107348
27 → A 28 A 0.008189	28 → C 29 C 0.000303	28 → C 29 G 0.367146	28 → C 29 U 0.004282
28 → C 29 A 0.000461	28 → G 29 C 0.090218	28 → G 29 G 0.000308	28 → G 29 U 0.014966
28 → G 29 A 0.000282	28 → U 29 C 0.000360	28 → U 29 G 0.018516	28 → U 29 U 0.006506
28 → U 29 A 0.317813	28 → A 29 C 0.000545	28 → A 29 G 0.000287	28 → A 29 U 0.177627
28 → A 29 A 0.000380	29 → C 30 C 0.000303	29 → C 30 G 0.173881	29 → C 30 U 0.000378
29 → C 30 A 0.000461	29 → G 30 C 0.396708	29 → G 30 G 0.002260	29 → G 30 U 0.007157
29 → G 30 A 0.000282	29 → U 30 C 0.000360	29 → U 30 G 0.002899	29 → U 30 U 0.000650
29 → U 30 A 0.208492	29 → A 30 C 0.002497	29 → A 30 G 0.000287	29 → A 30 U 0.201052
29 → A 30 A 0.002333	30 → C 31 C 0.000303	30 → C 31 G 0.209020	30 → C 31 U 0.002330
30 → C 31 A 0.002413	30 → G 31 C 0.662203	30 → G 31 G 0.000308	30 → G 31 U 0.036439
30 → G 31 A 0.000282	30 → U 31 C 0.000360	30 → U 31 G 0.012660	30 → U 31 U 0.002602
30 → U 31 A 0.021084	30 → A 31 C 0.000545	30 → A 31 G 0.000287	30 → A 31 U 0.048783
30 → A 31 A 0.000380	31 → C 32 C 0.000303	31 → C 32 G 0.267585	31 → C 32 U 0.004282
31 → C 32 A 0.000461	31 → G 32 C 0.187827	31 → G 32 G 0.000308	31 → G 32 U 0.007157
31 → G 32 A 0.000282	31 → U 32 C 0.002313	31 → U 32 G 0.032181	31 → U 32 U 0.008458
31 → U 32 A 0.097218	31 → A 32 C 0.000545	31 → A 32 G 0.000287	31 → A 32 U 0.390412
31 → A 32 A 0.000380	32 → C 33 0.666561	32 → G 33 0.002711	32 → U 33 0.309951
32 → A 33 0.020776	33 → C 34 0.012634	33 → G 34 0.000718	33 → U 34 0.985809
33 → A 34 0.000839	34 → C 35 0.233932	34 → G 35 0.331669	34 → U 35 0.359793
34 → A 35 0.074605	35 → C 36 0.172128	35 → G 36 0.220023	35 → U 36 0.294001
35 → A 36 0.313848	36 → C 37 0.231939	36 → G 37 0.226004	36 → U 37 0.270077
36 → A 37 0.271980	37 → C 38 0.000671	37 → G 38 0.237966	37 → U 38 0.002924
37 → A 38 0.758438	38 → C 0.166147	38 → G 0.012680	38 → U 0.122545
38 → A 0.698628	39 → C 90 0.003007	39 → G 90 0.000752	39 → U 90 0.047358

39 → A 90	0.000881	39 → 40	0.007089	39 → C 40	0.139147	39 → G 40	0.182203
39 → U 40	0.180513	39 → A 40	0.439051	40 → C 91	0.000712	40 → G 91	0.000978
40 → U 91	0.001084	40 → A 91	0.001035	40 → 41	0.976065	40 → C 41	0.003034
40 → G 41	0.001302	40 → U 41	0.001442	40 → A 41	0.014348	41 → C 92	0.000713
41 → G 92	0.000977	41 → U 92	0.001088	41 → A 92	0.001030	41 → 42	0.977563
41 → C 42	0.003038	41 → G 42	0.001294	41 → U 42	0.001435	41 → A 42	0.012862
42 → C 93	0.001003	42 → G 93	0.053461	42 → U 93	0.003357	42 → A 93	0.011759
42 → 43	0.004953	42 → C 43	0.026339	42 → G 43	0.556403	42 → U 43	0.179120
42 → A 43	0.163605	43 → C 94	0.003156	43 → G 94	0.002509	43 → U 94	0.010361
43 → A 94	0.008110	43 → 44	0.002968	43 → C 44	0.008442	43 → G 44	0.663010
43 → U 44	0.060786	43 → A 44	0.240658	44 → C 95	0.004842	44 → G 95	0.000898
44 → U 95	0.010156	44 → A 95	0.000944	44 → 45	0.975330	44 → C 45	0.002594
44 → G 45	0.001113	44 → U 45	0.002839	44 → A 45	0.001284	45 → C 96	0.010055
45 → G 96	0.027950	45 → U 96	0.002756	45 → A 96	0.001831	45 → 46	0.311847
45 → C 46	0.063857	45 → G 46	0.006889	45 → U 46	0.566127	45 → A 46	0.008688
46 → C	0.724378	46 → G	0.008692	46 → U	0.234192	46 → A	0.032738
63 → C 64 C	0.000303	63 → C 64 G	0.224638	63 → C 64 U	0.000378	63 → C 64 A	0.004365
63 → G 64 C	0.383043	63 → G 64 G	0.000308	63 → G 64 U	0.143809	63 → G 64 A	0.000282
63 → U 64 C	0.004265	63 → U 64 G	0.010708	63 → U 64 U	0.004554	63 → U 64 A	0.015228
63 → A 64 C	0.000545	63 → A 64 G	0.002239	63 → A 64 U	0.203004	63 → A 64 A	0.002333
64 → C 65 C	0.002255	64 → C 65 G	0.365193	64 → C 65 U	0.002330	64 → C 65 A	0.000461
64 → G 65 C	0.129262	64 → G 65 G	0.000308	64 → G 65 U	0.057913	64 → G 65 A	0.000282
64 → U 65 C	0.004265	64 → U 65 G	0.038038	64 → U 65 U	0.012362	64 → U 65 A	0.249487
64 → A 65 C	0.002497	64 → A 65 G	0.000287	64 → A 65 U	0.134679	64 → A 65 A	0.000380
65 → C 66 C	0.000303						
65 → C 66 G	0.166073	65 → C 66 U	0.002330	65 → C 66 A	0.000461	65 → G 66 C	0.416230
65 → G 66 G	0.002260	65 → G 66 U	0.036439	65 → G 66 A	0.000282	65 → U 66 C	0.000360
65 → U 66 G	0.043894	65 → U 66 U	0.004554	65 → U 66 A	0.101123	65 → A 66 C	0.002497
65 → A 66 G	0.000287	65 → A 66 U	0.222526	65 → A 66 A	0.000380	66 → C 67 C	0.000303
66 → C 67 G	0.015756	66 → C 67 U	0.002330	66 → C 67 A	0.000461	66 → G 67 C	0.794950
66 → G 67 G	0.000308	66 → G 67 U	0.007157	66 → G 67 A	0.000282	66 → U 67 C	0.000360
66 → U 67 G	0.002899	66 → U 67 U	0.000650	66 → U 67 A	0.024988	66 → A 67 C	0.004450
66 → A 67 G	0.000287	66 → A 67 U	0.144440	66 → A 67 A	0.000380	67 → C 68 C	0.000303
67 → C 68 G	0.004043	67 → C 68 U	0.000378	67 → C 68 A	0.000461	67 → G 68 C	0.955027
67 → G 68 G	0.000308	67 → G 68 U	0.005205	67 → G 68 A	0.000282	67 → U 68 C	0.000360
67 → U 68 G	0.000947	67 → U 68 U	0.000650	67 → U 68 A	0.003515	67 → A 68 C	0.004450
67 → A 68 G	0.000287	67 → A 68 U	0.023405	67 → A 68 A	0.000380	68 → C 69	0.002665
68 → G 69	0.000718	68 → U 69	0.963879	68 → A 69	0.032738	69 → C 70	0.006653
69 → G 70	0.020654	69 → U 70	0.971854	69 → A 70	0.000839	70 → C 71	0.979569
70 → G 71	0.000718	70 → U 71	0.008905	70 → A 71	0.010808	71 → C 72	0.002665
71 → G 72	0.608791	71 → U 72	0.000931	71 → A 72	0.387613	72 → C 73	0.002665
72 → G 73	0.000718	72 → U 73	0.000931	72 → A 73	0.995687	73 → C 74	0.044532
73 → G 74	0.224010	73 → U 74	0.242166	73 → A 74	0.489291	74 → C	0.138236

74 → G	0.008692	74 → U	0.816346	74 → A	0.036725	75 → C	0.052507
75 → G	0.243947	75 → U	0.102608	75 → A	0.600937	81 → C 81	0.029213
81 → G 81	0.031543	81 → U 81	0.041903	81 → A 81	0.119663	81 → C 16	0.029291
81 → G 16	0.034434	81 → U 16	0.282733	81 → A 16	0.431220	82 → C 82	0.069027
82 → G 82	0.009900	82 → U 82	0.022805	82 → A 82	0.044885	82 → C 17	0.723150
82 → G 17	0.012220	82 → U 17	0.095479	82 → A 17	0.022534	83 → C 83	0.029567
83 → G 83	0.008472	83 → U 83	0.024856	83 → A 83	0.011430	83 → C 18	0.658858
83 → G 18	0.010576	83 → U 18	0.220064	83 → A 18	0.036177	84 → C 84	0.016460
84 → G 84	0.017457	84 → U 84	0.023082	84 → A 84	0.020501	84 → C 19	0.579722
84 → G 19	0.036755	84 → U 19	0.235993	84 → A 19	0.070030	85 → C 85	0.065638
85 → G 85	0.083086	85 → U 85	0.103787	85 → A 85	0.094498	85 → C 20	0.066065
85 → G 20	0.264483	85 → U 20	0.200870	85 → A 20	0.121572	86 → C 86	0.019532
86 → G 86	0.002946	86 → U 86	0.011943	86 → A 86	0.022555	86 → C 21	0.197352
86 → G 21	0.008580	86 → U 21	0.492268	86 → A 21	0.244825	87 → C 87	0.070226
87 → G 87	0.001941	87 → U 87	0.013935	87 → A 87	0.010024	87 → C 22	0.101778
87 → G 22	0.090795	87 → U 22	0.491867	87 → A 22	0.219434	90 → C 90	0.321271
90 → G 90	0.152056	90 → U 90	0.227611	90 → A 90	0.165586	90 → C 40	0.003308
90 → G 40	0.115297	90 → U 40	0.003828	90 → A 40	0.011043	91 → C 91	0.113235
91 → G 91	0.119579	91 → U 91	0.151131	91 → A 91	0.141788	91 → C 41	0.100018
91 → G 41	0.109669	91 → U 41	0.127369	91 → A 41	0.137210	92 → C 92	0.112514
92 → G 92	0.119947	92 → U 92	0.150505	92 → A 92	0.142353	92 → C 42	0.100392
92 → G 42	0.109585	92 → U 42	0.127151	92 → A 42	0.137553	93 → C 93	0.146966
93 → G 93	0.179996	93 → U 93	0.258171	93 → A 93	0.301805	93 → C 43	0.064284
93 → G 43	0.001423	93 → U 43	0.045531	93 → A 43	0.001823	94 → C 94	0.130862
94 → G 94	0.148002	94 → U 94	0.448381	94 → A 94	0.155554	94 → C 44	0.008299
94 → G 44	0.005484	94 → U 44	0.014983	94 → A 44	0.088435	95 → C 95	0.286602
95 → G 95	0.296465	95 → U 95	0.043000	95 → A 95	0.253582	95 → C 45	0.034294
95 → G 45	0.037659	95 → U 45	0.039676	95 → A 45	0.008721	96 → C 96	0.401537
96 → G 96	0.143528	96 → U 96	0.255206	96 → A 96	0.069289	96 → C 46	0.003339
96 → G 46	0.104709	96 → U 46	0.014544	96 → A 46	0.007847		