RNA, SCFGs and Classifiers

Zsuzsanna Sükösd, Paula Tataru
Supervisor: Jotun Hein

University of Aarhus, September-October 2009

Table of Contents

Abstract
We developed, implemented and tested a stochastic context-free grammar (SCFG)-based method to classify RNA molecules into structural classes, by training grammars to simultaneously recognize certain types of RNAs and disrecognize other types. We tested our program using datasets obtained from thermodynamic (RNAfold) predictions of structures for 6000 tRNA and 6000 miRNA sequences, and we believe our program serves as a "proof of principle" of the structural classification of RNAs. We expect that incorporating evolutionary information from sequence alignments would significantly improve structure prediction. Furthermore, our method can naturally be extended to involve a higher number of grammars.

Introduction
In recent years, considerable interest has developed to elucidate the relationships between RNA sequence, structure and function, especially with regards to newly discovered roles of RNAs in cells and viruses. The computational prediction of RNA structures plays a vital role in bridging the gap between sequencing and experimental structure determination, and also facilitates the interpretation of experimental data.

In 1994, stochastic context free grammars (SCFGs) were introduced in two papers (Durbin and Eddy, 1994 and Sakikabara et al., 1994) to describe and analyze RNA structures. Knudsen and Hein (1999, 2003) coupled a SCFG to molecular evolution and thus created a comparative predictor. Most RNA gene predictors try to classify a sequence into two classes: "RNA gene" and "background functionless DNA". However, it is often of interest to classify an RNA gene/structure into functional classes. Typically, this is either done by homology (if two sequences are similar, they probably have the same function) or by some additional classifier algorithm (Batuwita and Palade, 2009 & Klingelhoefer, Moutsianas and Holmes, 2009).

In this project, we attempted to classify RNA molecules into structural classes, by training SCFGs to recognize some types of RNAs at the same time as disrecognizing others.

Our stochastic context-free grammar
RNA sequences are long chains of 4 nucleotides, and can be represented as a string consisting of 4 different letters: A (adenine), C (cytosine), G (guanine) and U (uracil). RNA secondary structure (ie. the two-dimensional folding of RNA molecules) can also be described as a string of symbols: the single-nucleotide symbol ".", and the base-pairing symbols " ( ) " , where the brackets indicate that the nucleotide in position of the first one base-pairs with the nucleotide in the position of the second one. (Figure 1a) This description makes it possible to produce RNA structures using context-free grammars.

A formal grammar in language theory consists of:

- a set of nonterminal symbols
- a set of terminal symbols (disjoint from the set of nonterminals)
- a set of production rules that map one string of symbols to another.

A distinguished nonterminal symbol is the start symbol.

A context-free grammar is a grammar in which every production rule maps a single nonterminal symbol into a string of terminal and nonterminal symbols. In this project we used a context-free grammar designed by Bjarne Knudsen that can generate strings to describe all RNA structures that contain no pseudoknots and have a minimum of 2 nucleotides in between any base-pairs. By assigning a probability to the different rules that generate strings from the same nonterminal symbol, we obtain the stochastic context-free grammar:

S -> LS	prob. p_1	L -> .	prob. $1 - p_2$
S -> L	prob. $1 - p_1$	F -> (F)	prob. p_3
L -> (F)	prob. p_2	F -> LS	prob. $1 - p_3$

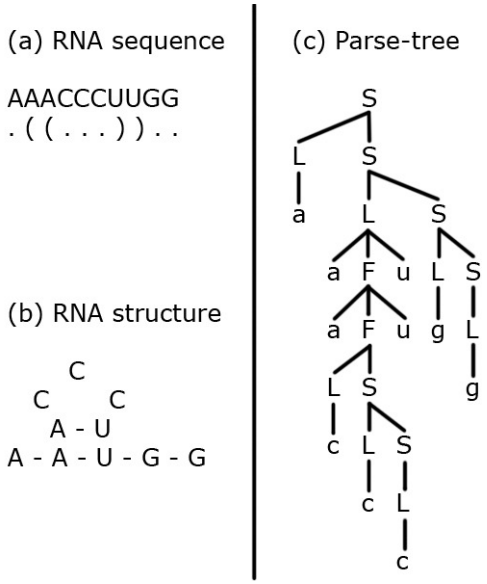


Figure 1: Parsing RNA secondary structure using a SCFG.

(Figure dapted from "Final Project: RNA Structure Prediction Using SCFG - 2008")

For every string produced by the grammar, a parse tree (an ordered, rooted tree) can be drawn that describes which rules were used and in what order to generate the final string. The internal nodes of the tree are the nonterminals of the grammar; the leaf nodes are terminal symbols (Figure 1c).

For each parse-tree, we can define the probability of the structure *s* (a string over the alphabet { "." , "(" , ")" }) given a SCFG grammar *G*, *P(s|G)*, as the product of the probabilities of all the rules in the grammar that were used to generate the structure:

Definition 1: $$P(s|G) = \prod_{\text{rules used to generate } s} P(\text{rule})$$

For example, the probability of the structure given grammar for the parse-tree in Figure 1c is:

$$P(s = .((\ldots))\ldots |G) = p_1{}^2(1-p_1)^4 p_2(1-p_2)^6 p_3(1-p_3)$$

This treatment does not consider the fact that different nucleotides can have different base-pairing properties; a final probability allocated to a structure should be sequence-dependent if the sequence is known. Furthermore, in our classification problem we wish to determine if a sequence is best parsed by a particular grammar, and it is therefore of interest to define the probability of a sequence given the grammar.

We therefore first define the probability of structure *s* given sequence *x* (a string over the alphabet {"a", "c", "g", "u"}) as the product of the probabilities of single and dinucleotides that appear in the structure s of that sequence:

Definition 2:

$$P(s|x) = \left[\prod_{x \in \text{single nucleotides}} p_x{}^{\text{no. of occurence}}\right]\left[\prod_{yz \in \text{dinucleotides}} p_{yz}{}^{\text{no. of occurence}}\right]$$

where $\sum_{x \in \text{single nucleotides}} p_x = 1$ and $\sum_{yz \in \text{dinucleotides}} p_{yz} = 1$

The probabilities of single and dinucleotides are parameters of our distribution and are elements of a 4x1 and a 4x4 matrix, respectively.

For example, $P(s = (\ldots) \,|\, x = \text{acgcu}) = p_{au} \cdot p_c{}^2 \cdot p_g$

We now define the probability of a structure *s* given sequence *x* and grammar *G*:

Definition 3: $$P(s|x,G) = P(s|x) \cdot P(s|G)$$

This expression incorporates both the probabilities in the grammar and the probabilities of single- and dinucleotides as parameters, ie. *P(s|x,G)* is a function of (4-1)+(4-1)+(4x4-1)=21 parameters.

Using this, we finally define the probability of a sequence *x* given grammar *G* as a product over the probabilities of all structures possible for that sequence under the grammar:

Definition 4: $$P(x|G) = \sum_{s,\ |s|=|x|} P(s|x,G) = \sum_{s,\ |s|=|x|} P(s|x) \cdot P(s|G)$$

Effectively, this expression gives the total probability that the sequence can be parsed to a structure using our SCFG at all.

The CYK and inside algorithms

The pseudo code for both the CYK and inside algorithms in our implementation is found the Appendix A. Here we only give a general overview of how the algorithms function.

The Cocke-Younger-Kasami (CYK) algorithm is a dynamic programming algorithm that determines whether a string can be generated by a context-free grammar, and if yes, how. This is known as parsing the string. The standard CYK algorithm works for grammars that are in the CNF (Chomsky Normal Form). The SCFG grammar described here is not in CNF, therefore we adapted the CYK algorithm to this particular grammar. The CYK algorithm is recursive. (Figure 2)

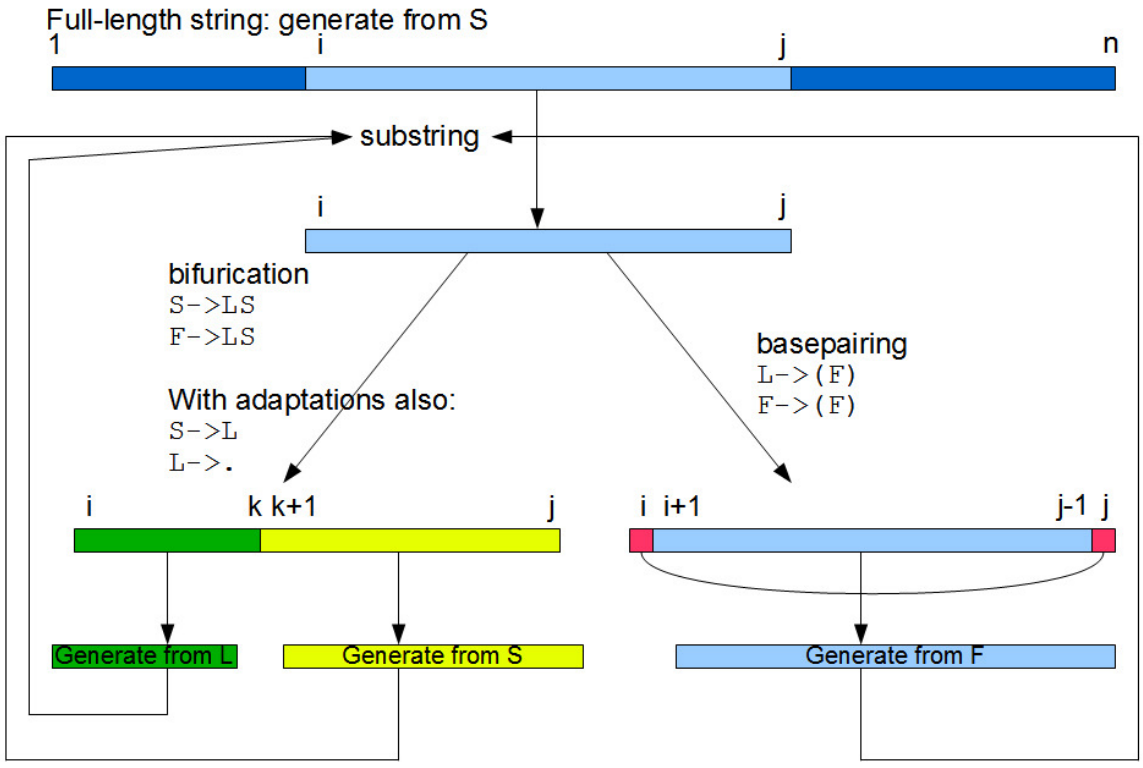


Figure 2: Illustration of the modified CYK recursion for our grammar.

In the standard CYK algorithm, we ask, for each substring: "Can this substring be generated from the given nonterminal symbol (S, L or F)?" The full-length string can be generated from the nonterminal symbol S if and only if there is a recursive partitioning of the sequence such that all substrings (and substrings of substrings, etc.) can be generated by the grammar. RNA structures in practice can always be generated by the rules of this grammar unless they have pseudoknots (base-pairing from inside a loop to somewhere outside the loop), or fewer than two single bases in a loop (because the smallest number of bases that can be generated from the non-terminal F is 2).

For SCFG grammars, the CYK algorithm can be adapted so instead of a TRUE/FALSE result it gives the probability of a given structure (Definition 1). By choosing the parsing with the highest probability in each iteration step, we can obtain the structure with the highest probability (the *s* for which *P(s|G)* is highest). If sequence information is additionally known, the structure with the highest *P(s|x,G)* can be calculated in a straightforward manner (Definition 3).

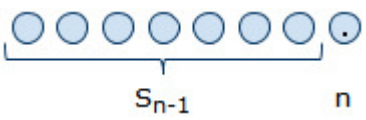
A modification of the CYK algorithm (the so-called "inside algorithm") can be used to find *P(x|G)* (Definition 4). In the inside algorithm, instead of choosing the structure with the highest probability, the sum of probabilities for all substructures is calculated.

Results and discussion

Investigating the space search

To investigate in what space the CYK algorithm needs to search for the best structure, we determined by recursion the number of structures, S_n ,our grammar could generate as a function of structure length *n*. Consider S_n as a function of S_k, for $k < n$:

To obtain a structure of length *n* starting from a string of length *n*-1, there are two choices:



addition of ".":

addition of ")" paired with some "(":

Because any hairpin loops must have a length of at least 2 bases, *n*-*k*-1 ≥ 2.

Hence we obtain the following recursion: $$S_n = S_{n-1} + \sum_{k=1}^{n-3} S_{k-1} S_{n-k-1}$$

with $S_0 = S_1 = S_2 = S_3 = 1$.

As shown by Nkwanta , 2006, $$S_n = \sum_{k \geq 1} \frac{1}{n-k} \cdot \binom{n-k}{k} \cdot \binom{n-k}{k-1}$$

We can also consider the sum of the probabilities, $P_G(n)$, of all structures possible for a given structure length *n*: $$P_G(n) = \sum_{s, |s|=n} P(s \mid G)$$

$P_G(n)$ can serve as a normalization constant when we wish to compare probabilities for predicted structures for sequences of a particular length.

One could attempt to derive a general formula for $P_G(n)$, according to the grammar's rules probabilities. We, instead, used our program to show the general shape of $P_G(n)$ for different probabilities in our grammar (Figures 3 and 4).

Grammar 1	**(G1)**	**Grammar 2**	**(G2)**	**Grammar 3**	**(G3)**	**Grammar 4**	**(G4)**
Rule	*Prob.*	*Rule*	*Prob.*	*Rule*	*Prob.*	*Rule*	*Prob.*
S -> LS	0.5	S -> LS	0.4	S -> LS	0.7	S -> LS	0.2
S -> L	0.5	S -> L	0.6	S -> L	0.3	S -> L	0.8
L -> (F)	0.5	L -> (F)	0.4	L -> (F)	0.1	L -> (F)	0.2
L -> .	0.5	L -> .	0.6	L -> .	0.9	L -> .	0.8
F -> (F)	0.5	F -> (F)	0.4	F -> (F)	0.1	F -> (F)	0.2
F -> LS	0.5	F -> LS	0.6	F -> LS	0.9	F -> LS	0.8

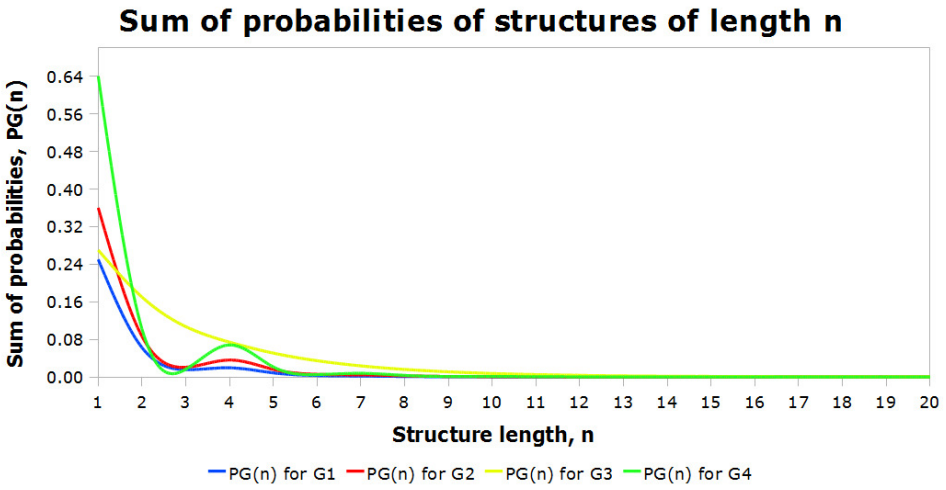


Figure 3: $P_G(n)$ for the four analyzed grammars.

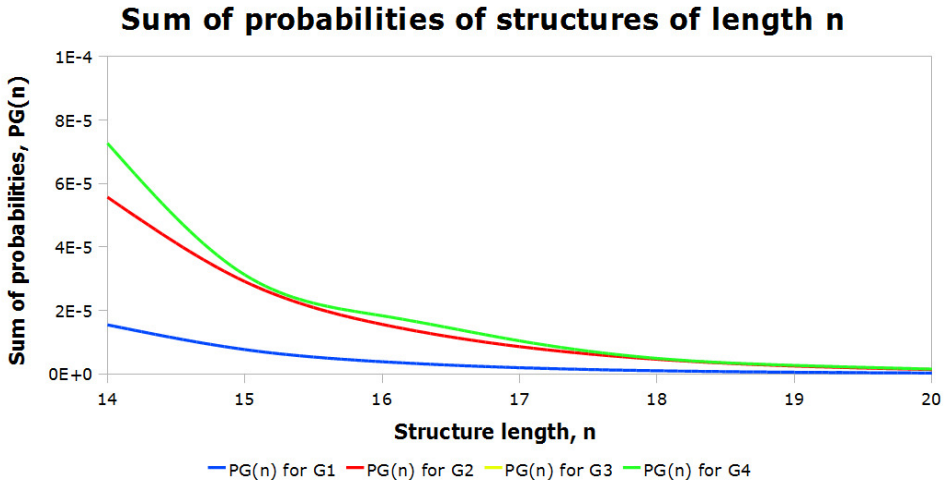


Figure 4: $P_G(n)$ for the four analyzed grammars - zooming in to n=14..20. The yellow curve (for G3) is not visible at this zoom level, because its probability values are too large.

In all cases, the structure of length 1 (ie. the single unpaired nucleotide) has the highest probability, and the general pattern is that longer structures have smaller total probabilities. This is because by the addition of nucleotides, additional rules need to be used, increasing the number of probabilities to be multiplied together, decreasing their product. An exception to this is n = 4: base-pairing first becomes possible with 4 nucleotides,

ie. to use the rule L->(F). In the limit where structure length approaches infinity, all grammars predict zero probability.

Training the grammars

To "train" a grammar, we wish to find the parameter values that maximize the probability of our input data. Depending on the problem, the data can be:

- one set of structures to recognize
- two sets of structures ("type A" and "type B"), one to recognize and one to disrecognize
- one set of sequences and corresponding structures to recognize
- two sets of sequences ("type A" and "type B" and corresponding structures, one to recognize and one to disrecognize.

In the followings, we consider the model on each category individually.

One set of structures ("independent" training)

We wish to train one grammar to recognize one set of structures, ie. to maximize $P(s_1,...,s_n|G)$ for input structures $s_1,...,s_n$. (If given another set of structures to recognize by another grammar, the probabilities for the second grammar can be determined independently of the first set of structures.) Since the input structures are independent,

$$\max P(s_1,...,s_n|G) = \max \prod_{i=1}^{n} P(s_i|G)$$

According to Definition 1, this expression will have the following form:

$$P(s_1,...,s_n|G) = \prod_{\text{rules}} {p_{rule}}^{f_{rule}}$$
$$= p_1^{f_1}\cdot(1-p_1)^{f_2}\cdot p_2^{f_3}\cdot(1-p_2)^{f_4}\cdot p_3^{f_5}\cdot(1-p_3)^{f_6}$$

where $f_1..f_6$ are now the total number of times each rule was used in the parsing of all input structures.

Maximizing this expression with respect to $p_1...p_3$ is straightforward and yields:

$$p_1 = \frac{f_1}{f_1+f_2}\ ,\ p_2 = \frac{f_3}{f_3+f_4}\ ,\ p_3 = \frac{f_5}{f_5+f_6}$$

Two sets of structures

Here we consider the situation where we train a grammar to recognize "type A" structures and disrecognize "type B" structures. (The "complementary" grammar will recognize "type B" structures and disrecognize "type A" structures.) The input structures are assumed to be independent, as before. Our parameter space is identical to what we had in the independent training with structures. The function to maximize, however, is not straightforward. We have tested two different functions, of the following form:

"Minus" function $\prod_{\text{rules}} {p_{rule}}^{f_{rule}} - \prod_{\text{rules}} {p_{rule}}^{g_{rule}}$

where f_{rule} are the numbers of times the corresponding rules were used to parse the "good" input structures (the structures to recognize), and g_{rules} are the numbers of times the corresponding rules were used to parse the "bad" input structures (the structures to disrecognize).

"Times 1 minus" function $\prod_{\text{rules}} {p_{rule}}^{f_{rule}} \cdot \left[1 - \prod_{\text{rules}} {p_{rule}}^{g_{rule}}\right]$

with the notation as in the "minus" model.

We used numerical routines (NLP command) in the Optimization package of Maple to maximize these functions (we determined f_{rule} and g_{rule} from our datasets), specifying that all parameters are defined only in the real domain and over the interval [0, 1).

Testing of models with only structures as input

Once the parameters of the two grammars were determined for all three "structure-only" models ("independent", "minus", "times 1 minus"), we tested the ability of our program to train two grammars and predict the type of arbitrary input structures.

First test

First training dataset ("good" structures for grammar A = "bad' structures for grammar B):

`3 x (((.....)))` `7 x ((((....))))`

Second training dataset ("good" structures for grammar B = "bad" structures for grammar A):

`5 x ((...)(...))` `5 x ((..))((..))`

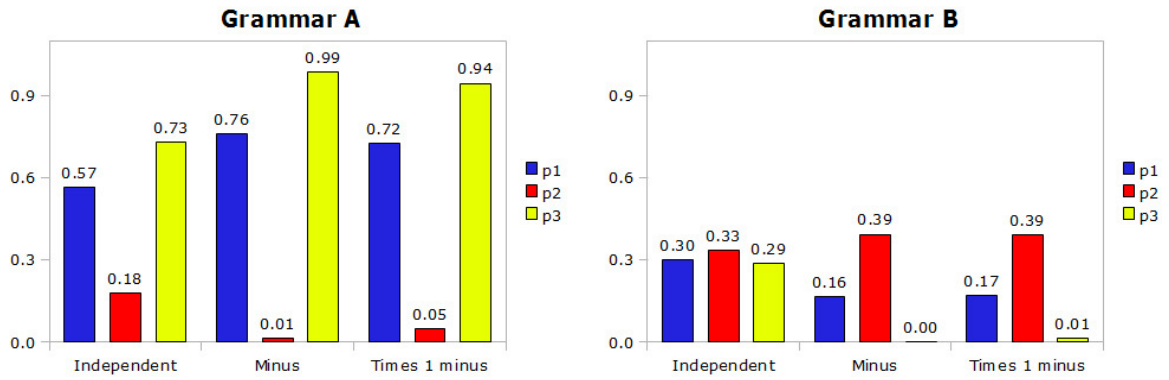
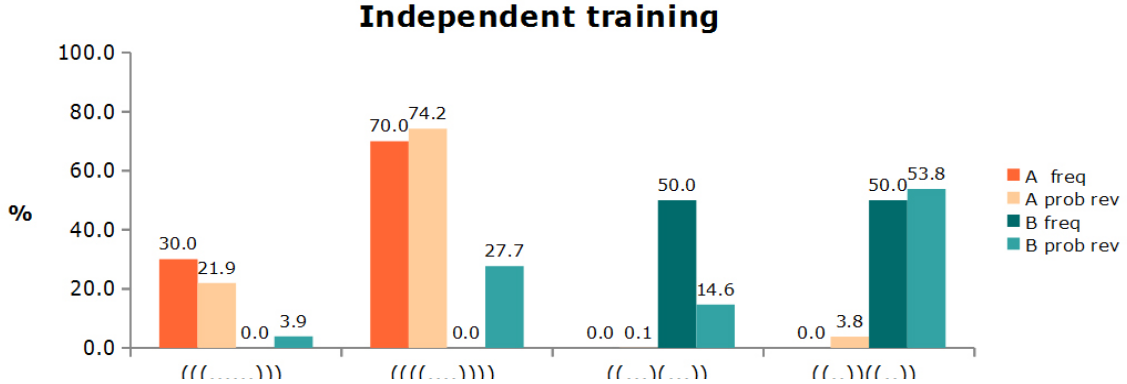
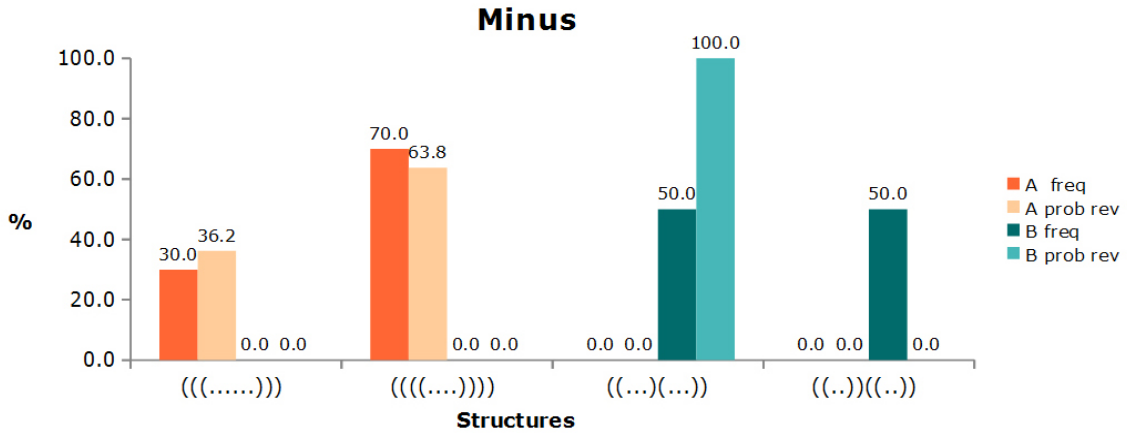


Figure 5: Results of the training of grammars with the different models in the first test. p1-p3 are the probabilities in the grammars as determined by the indicated models.

a)



b)



c)

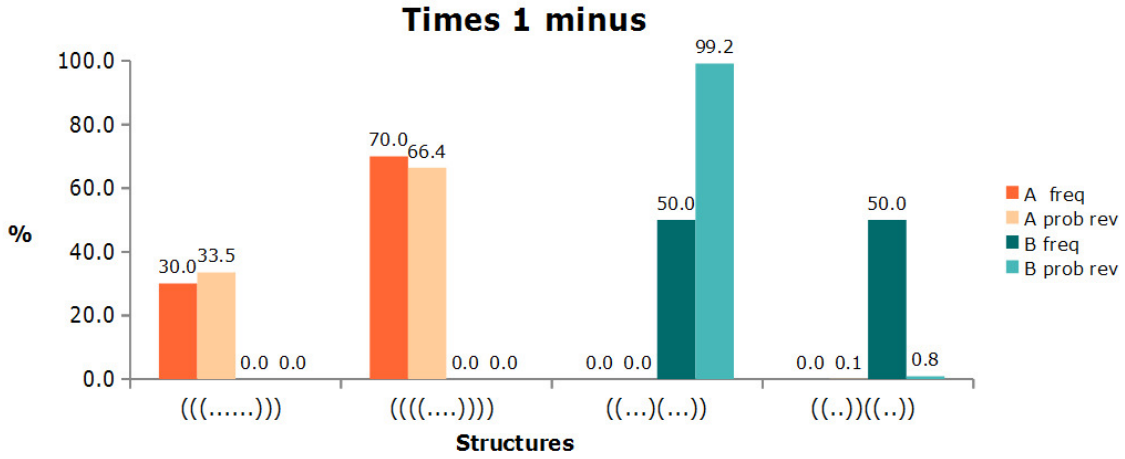


Figure 6: Results of the first test, for the independent (a), "minus" (b) and "times 1 minus" (c) models. "Afeq" and "Bfreq" are the fractional frequencies of structures in our training data. "A prob rev" and "B prob rev" are the probabilities predicted for the same structures by grammar A and grammar B, respectively, normalized such that the probabilities for the shown structures add up to 100% (so they are comparable to the frequency distribution).

Second test

First data set ("good" structures for grammar A, "bad" structures for grammar B)

`2 x (((((.........)))))` `2 x (((((.(...)..)))))` `6 x ((((((....))))))`

Second data set ("good" structures for grammar B, "bad" structures for grammar A)

`4 x ((..))(..)((..))` `3 x (.(..)(..)(..).)` `3 x ((..)..(..))(..)`

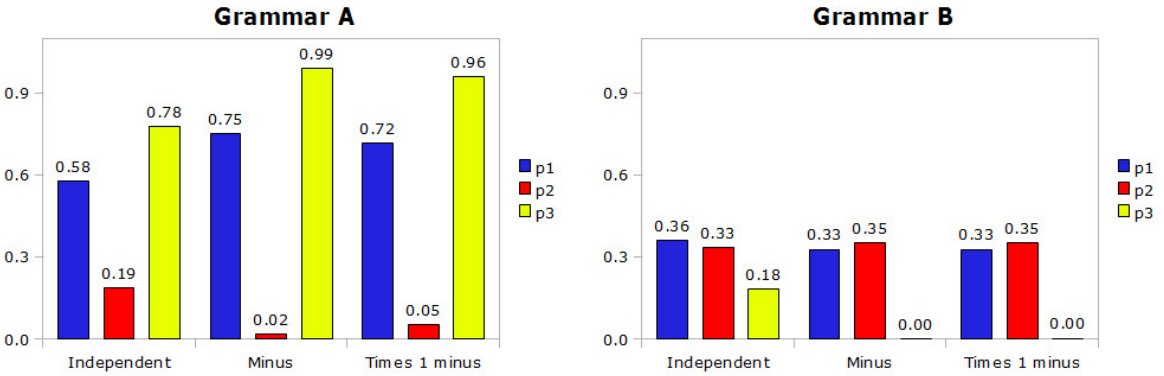
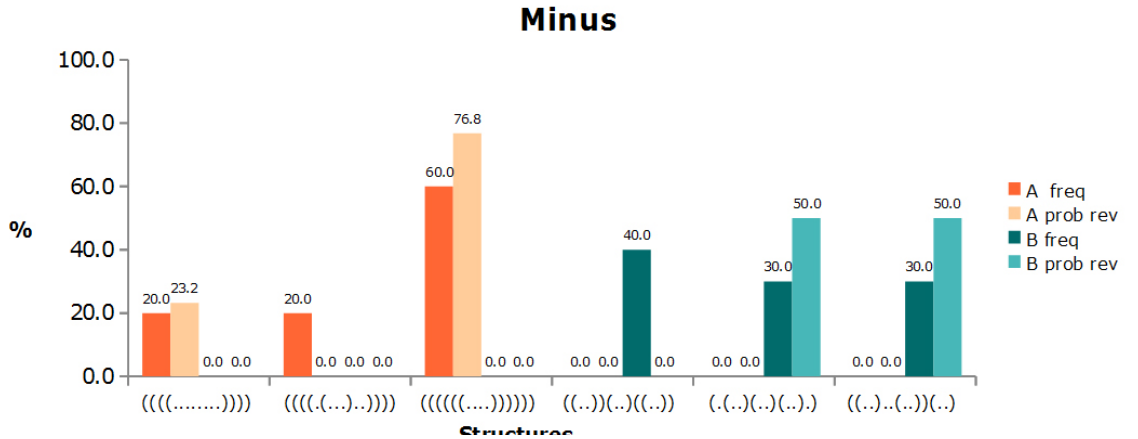


Figure 7: Results of the training of grammars with the different models in the second test. p1-p3 are the probabilities in the grammars as determined by the indicated models.

a)

b)



c)

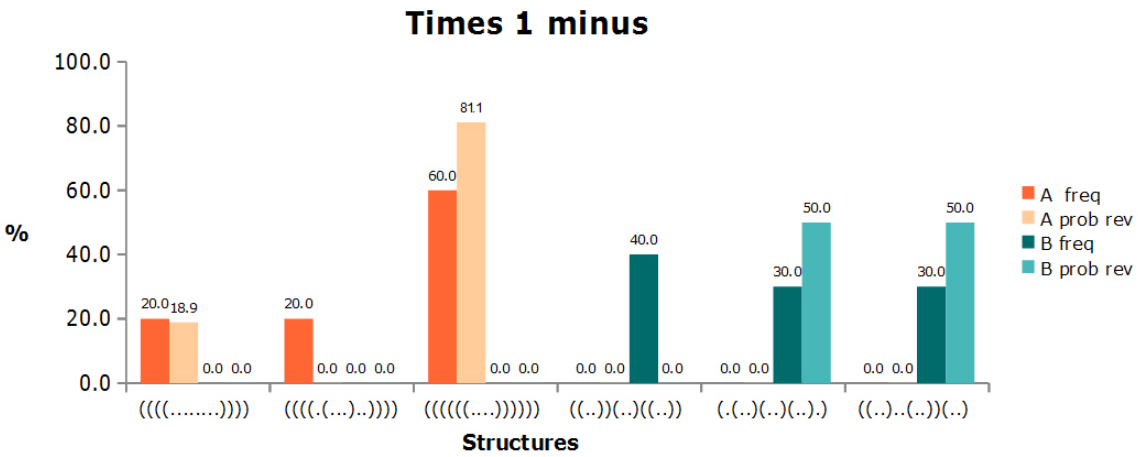


Figure 8: Results of the second test, for the independent (a), "minus" (b) and "times 1 minus" (c) models. "Afeq" and "Bfreq" are the fractional frequencies of structures in our training data. "A prob rev" and "B prob rev" are the probabilities predicted for the same structures by grammar A and grammar B, respectively, normalized such that the probabilities for the shown structures add up to 100% (so they are comparable to the frequency distribution).

Figures 5 and 7 demonstrate that both the "minus" and the "times 1 minus" models are more effective in separating the probabilities in the rules of the grammars than the independent model. In both tests (Figure 6 and 8), we can observe that independent training did not result minimizing the probabilities of "bad" structures. For example, independent resulted in a large number of type A structures having relatively high probabilities under grammar B. In contrast, both the "minus" and "times 1 minus" models minimized the recognition of type A structures by grammar B.

In both tests, when the grammars are trained together, one "good" structure for grammar B is actually predicted by the trained grammar B to have a probability of nearly 0 (<0.1%), while the independent model predicts a higher probability for the same structure. This is because the rule F -> (F) is used more often to parse the structures that are "bad" for grammar B than to parse those which are "good" for grammar B, and hence this rule is given a very small probability when the grammars were trained with both datasets, but not when they were trained with just the "good" dataset.

Our data is insufficient to decide with certainty whether the "minus" or the "times 1 minus" model is better for training the grammars together. We observed that the "times 1 minus" model appeared to map closer to the frequencies than the "minus" model in the first test, but not in the second test, where the "minus" model works slightly better for grammar A. However, we found that in the second test the "minus" model gave a higher probability for grammar A predicting a "bad" structure than the "times 1 minus" model did.

One set of sequences and corresponding structures

We now wish to train one grammar to recognize one set of structures given the sequences, ie. to maximize *P(s1,...,sn|x1...xn, G)* for input sequences *x1,...,xn* with corresponding structures *s1,...,sn*. If given another set of structures and sequences to recognize by another grammar, the probabilities for the second grammar can be determined independently of the first set of structures and sequences. Since the input structures and sequences are independent,

$$\max P(s_{1,\ldots,}s_n|x_{1,\ldots}x_n,G) = \max \prod_{i=1}^{n} P(s_i|x_i,G)$$

For each grammar, the expression to maximize will therefore be of the form:

$$\prod_{\text{rules}} {p_{rule}}^{f_{rule}} \prod_{x \in \text{single nucleotides}} {p_x}^{f_x} \prod_{yz \in \text{dinucleotides}} {p_{yz}}^{f_{yz}},$$

where:

- p_{rule} are the probabilities of the rules in the grammar (probabilities for rules from same nonterminal sum to 1, ie. 3 parameters in total) and f_{rule} are the numbers of times the corresponding rules were observed in the data
- p_x are the probabilities of single (unpaired) nucleotides (4x1 vector summing to 1, ie. 3 parameters), and f_x are the numbers of times they were observed in the data
- p_{yz} are the probabilities of dinucleotides (4x4 vector summing to 1, ie. 15 parameters), and f_{yz} are the numbers of times they were observed in the data.

Two sets of sequences and corresponding structures

The situation here is a mixture of "two sets of structures" and "one set of sequences and structures". Because of a lack of time, we decided only to test the "times 1 minus" model. In the future, other functions could also be tested.

"Times 1 minus" function

When training the grammars together, each grammar should have high probabilities for the "good" sequences and their structures and low probabilities for the "bad" sequences and their related structures. For each grammar, we wish to maximize the following expression:

$$\prod_{\text{rules}} {p_{rule}}^{f_{rule}} \prod_{x \in \text{single nucleotides}} {p_x}^{f_x} \prod_{yz \in \text{dinucleotides}} {p_{yz}}^{f_{yz}} \cdot$$
$$\left[1 - \prod_{\text{rules}} {p_{rule}}^{g_{rule}} \prod_{x \in \text{single nucleotides}} {p_x}^{g_x} \prod_{yz \in \text{dinucleotides}} {p_{yz}}^{g_{yz}}\right]$$

where:

- p_{rule}, p_x and p_{yz} are defined as before
- f_{rule} are the numbers of times the corresponding rules were observed in producing the structures of the "good" sequences and g_{rule} are the number of times the corresponding rules were observed in producing the structures of the "bad" sequences
- f_x are the numbers of times the single nucleotides were observed in the "good" data and g_x are the number of times they were observed in the "bad" data
- f_{yz} are the numbers of times the dinucleotides were observed in the "good" data and g_{yz} are the number of times they were observed in the "bad" data.

Testing of models with sequences and structures as input

To test the training of our grammars with sequence and structures as input, we used data from online RNA databases. We obtained 6000 miRNA sequences from miRBase and 6000 eukaryotic tRNA sequences from GtRNAdb and predicted their structures using the thermodynamic folding algorithm RNAfold. It is important to note that RNAfold does not necessarily predict the "real-life" structure of these molecules, but for the purposes of training our grammars RNAfold predictions are suitable.

We first ran our program to find the frequencies for the rules, single and dinucleotides, in both cases, using 3000 miRNA sequences and 3000 tRNA sequences. We then used the NLP command in the Optimization package of Maple to maximize the functions in both the "independent" and the "times 1 minus" models. The parameters are defined only in the real domain and over the interval [0, 1).

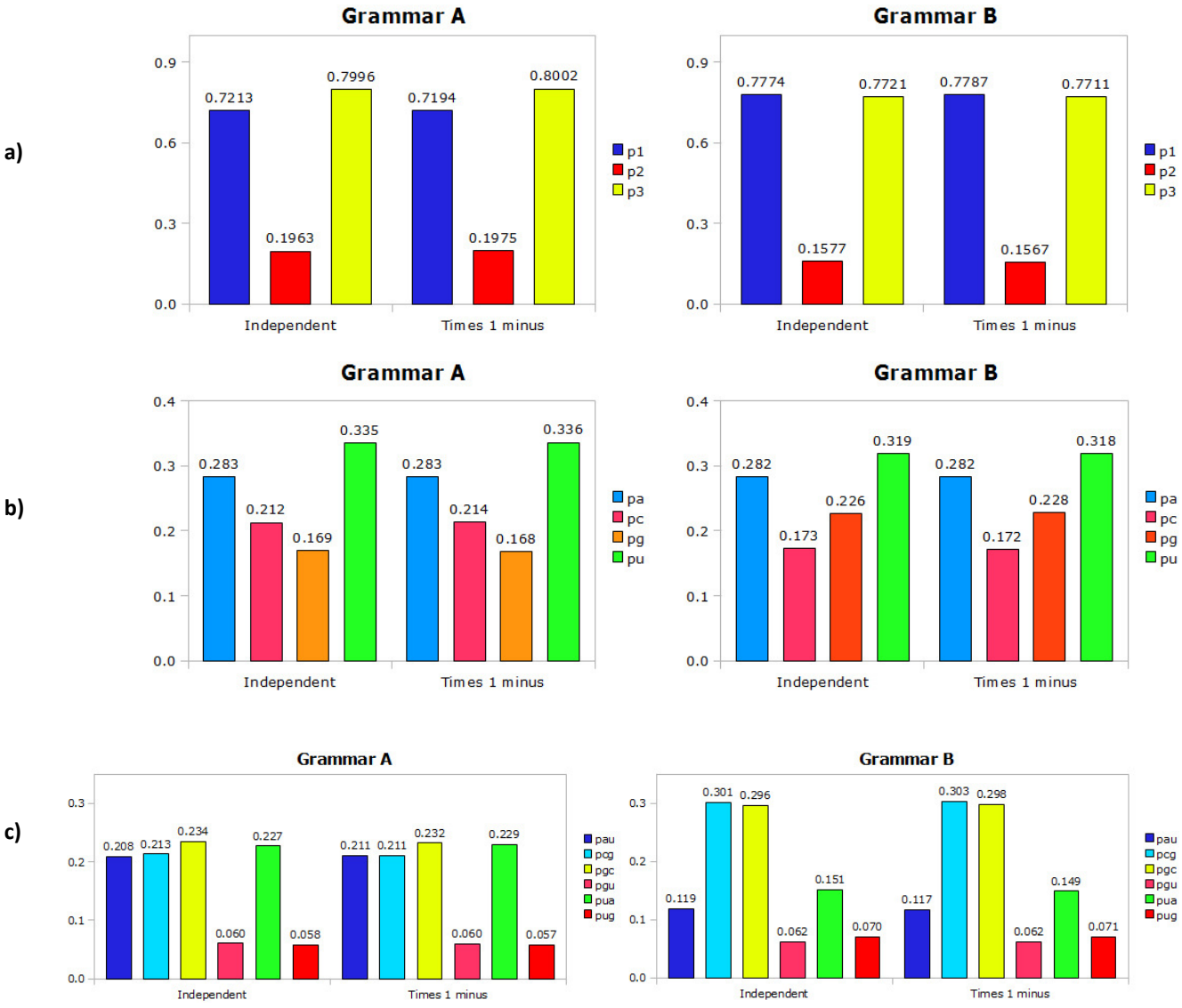


Figure 9: The probabilities obtained for the rules of the grammar (a), single nucleotides (b) and dinucleotides (c)

We observed that the independent and "times 1 minus" models led to very similar probabilities in the two grammars (Figure 9). We therefore expected the classification and structure prediction to produce similar results in the two cases.

For testing, we used:

- the "old" sequences, ie. the 3000 miRNAs and 3000 tRNAs that we already used in the training process
- "new" sequences, ie. the 3000 miRNAs and 3000 tRNAs that we didn't use in the training.

Testing classification

In order to classify a sequence *x*, we chose the grammar that gave the highest *P(x|G)* and the type of *x* would then be the type that the "winning" grammar was assigned to. The results of this testing are show in Figure 10.

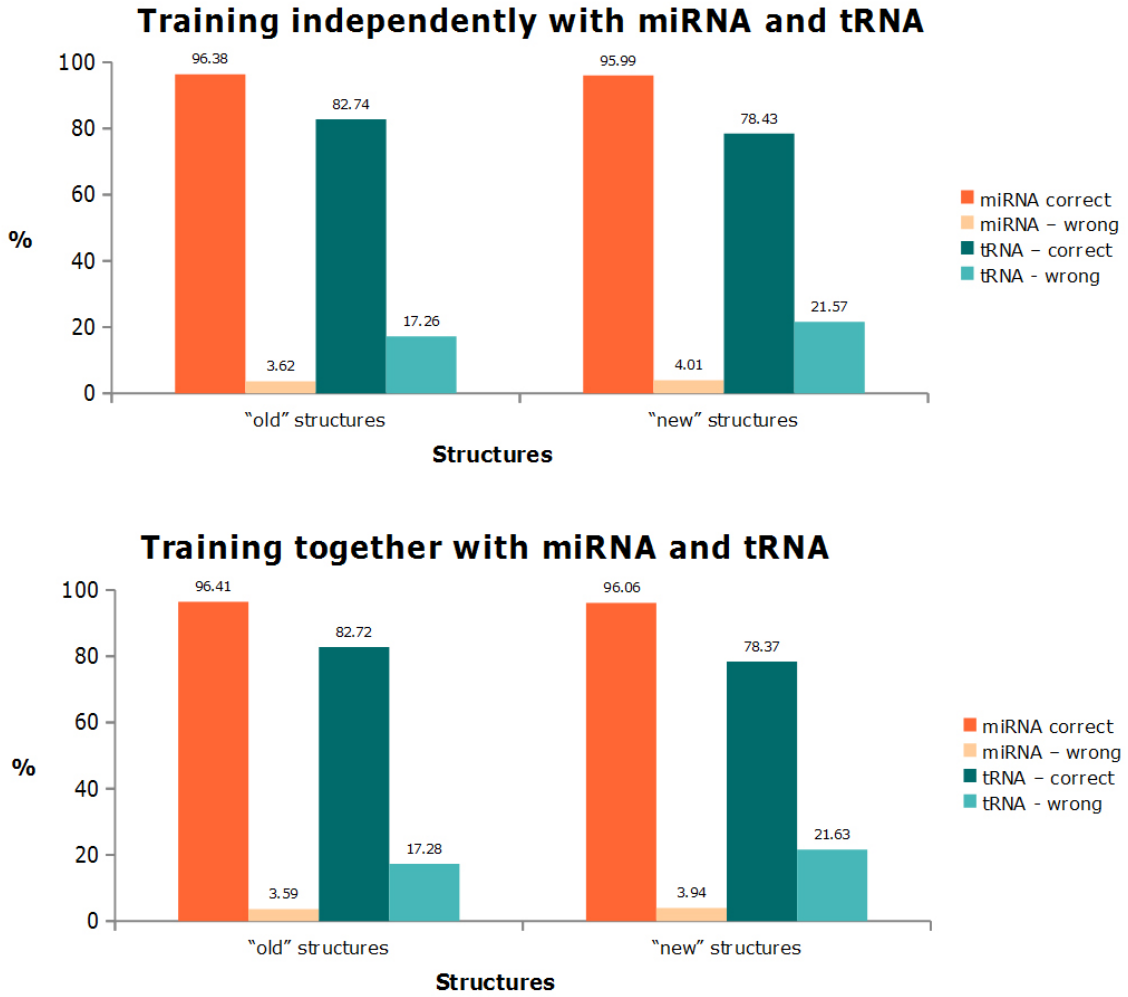


Figure 10: Classification results. Each bar represents the fraction of times our program determined correctly or incorrectly the nature of the type of RNA. The bars are normalized such that the sum of correct and incorrect predictions for each type of RNA ("old mRNA", "new mRNA", "old tRNA", "new tRNA") is 100%.

Figure 10 demonstrates that both the independently trained grammars and the grammars trained with the "times 1 minus" model were able to differentiate between tRNAs and miRNAs: this is true for both the sequences we trained them with ("old") and other, previously "unseen" sequences ("new"). Since miRNAs were recognized better by the "times 1 minus" model but tRNAs were recognized better by the independent training model, we cannot say from this data which model is more appropriate to distinguish RNAs.

As expected, the "old" structures were recognized better than the "new" structures in both cases, but the difference is small. Both the independent and the "times 1 minus" models are more effective at recognizing miRNAs correctly than tRNAs. We speculate that the reason for this may be that miRNA hairpins are generally longer than tRNAs.

Testing structure prediction

Since our structure prediction did not incorporate the use of evolutionary information, our grammars are not expected to predict structures well. Nevertheless, we attempted to compare the structure predictions of our grammars to that of RNAfold. It would be possible to use a metric for RNA secondary structures, and this way the "distance" between our predictions and the RNAfold predictions could be computed, for all 3000x4 = 12 000 sequences. For lack of time, we were not able to implement this, but we did create visual depictions of a few different structures using VARNA (http://varna.lri.fr/; see Appendix B for images).

In general, it appears that our trained grammars are quite efficient at predicting RNA structures even for previously unseen sequences. In a few cases, the structures predicted by our trained program actually appear more realistic than the predictions of RNAfold (based only on the single input sequence). It is interesting to note that the "independent" and "times 1 minus" models appear to predict the same best structure more often than not; the only exception we have observed is for a miRNA sequence (miRNA-2).

Final remarks

In this work, we have tested various models to classify RNAs structurally using a SCFG. We have thus been able to structurally classify tRNA and miRNA sequences with high accuracy after training two grammars specialized to recognize only one of them. We believe this serves as a "proof of concept" that structural classification of RNAs using SCFGs is possible and may be a useful approach in RNA secondary structure prediction. If we had had the time, we would also have tested whether the grammars trained with only one type of RNA were more able to predict structures than if they had been trained with a mixture of both types.

This introductory work could be improved and extended in many ways. We expect that by incorporating evolutionary information (ie. alignments), our structure predictions would be improved significantly. Not less importantly, our program is currently only able to distinguish two types of RNAs, and it "forces" any input RNA into one of those types. It would be important to introduce a statistical measure for how "sure" we can be that a particular structure actually falls into the category our program allocates it to, and define a new category of "unknown" RNA types. Our approach has a natural extension to an arbitrary number of RNA types to recognize.

Finally, it would be interesting to investigate how grammars of other forms perform in the classification procedure.

References

Dowell & Eddy. **Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction** BMC Bioinformatics, 5:71, 2004

Durbin et al. **Biological sequence analysis. Probabilistic models of proteins and nucleic acids** Cambridge University Press, 1998

Knudsen & Hein. **RNA secondary structure prediction using stochastic context-free grammars and evoluionary history** Bioinformatics vol. 15 no. 6, 1999

Knudsen & Hein. **Pfold: RNA Secondary Structure Prediction Using Stochastic Context-Free Grammars** NAR 31(13), 3423–342, 2003

Sakakibara et al. **Stochastic Context-Free Grammars for tRNA Modeling** NAR 22(23), 5112-, 1994

Schmitt & Waterman. **Linear trees and RNA secondary structure** Discrete Applied Mathematics 51 317-323, 1994

Waterman & Smith. **RNA secondary structure: A complete mathematical analysis** Mathematical Biosciences, vol.41, pp.257–266, 1978

Online references:

Final Project: RNA Structure Prediction Using SCFG - 2008
http://www.ece.tamu.edu/~bjyoon/ecen689-612-spring08/ecen689-612-final-project.pdf

Hein, J. **RNA, Stochastic Context Free Grammars and Classifiers - 2009**
http://www.stats.ox.ac.uk/__data/assets/pdf_file/0003/5268/rna_classifiers.pdf

Nkwanta, A. **Predicting RNA Secondary Structures: A Lattice Walk Approach to Modeling Sequences Within the HIV-1 RNA Structure - 2006**
dimacs.rutgers.edu/Workshops/Diseases/slides/nkwanta.ppt

miRNA database - miRBase
http://www.mirbase.org/ftp.shtml

tRNA database - Genomic tRNA Database
http://gtrnadb.ucsc.edu/download.html

RNAfold code - Vienna RNA Package - RNA Secondary Structure Prediction and Comparison
http://www.tbi.univie.ac.at/~ivo/RNA/

VARNA - Visualization Applet for RNA
http://varna.lri.fr/

Appendix A: The program pseudo code

Determine if a string can be generated by the grammar

Let S be nonterminal 1, L nonterminal 2 and F nonterminal 3.
Let the input be a string X consisting of n characters: x1 ... xn.
Let P[n,n,3] be an array of booleans: P[i,j,k] = true if it is possible that, starting from the nonterminal no. k we will obtain, after applying rules from the grammar, the substring xi....xi+j-1 that starts at position i and has j characters/length.

```
// initialize P with false
for i=1 to n
   for j=1 to n-i+1
      for k=1 to 3
         P[i,j,k] = false;

// S -> L and L -> . lead to:
for i=1 to n {
   P[i,1,2] = true;
   P[i,1,1] = true;
}
```

```
for j=2 to n //length of the substring
   for i=1 to n-j+1 {//start of the substring
      if ( xi=="(" and xi+j-1==")" and P[i+1,j-2,3] )
         then P[i,j,1] = P[i,j,2] = P[i,j,3] = true;
      for k=1 to j-1 { //partition of the substring
         if ( P[i,k,2] and P[i+k,j-k,1] )
            then P[i,j,1] = P[i,j,3] = true;
   }
```

if (P[1,n,1] == true) than X can be generated by the grammar G.

Determine the parse tree

Let the rules of the grammar have the following associted numbers:

No.	Rule	Probability
1	S->LS	p1
2	S -> L	1-p1
3	L -> (F)	p2
4	L -> .	1-p2
5	F -> (F)	p3
6	F -> LS	1-p3

Let parse_tree[n,n,3] be an array of strings: parse_tree[i,j,k] = the sequence of rules needed to generate xi...xi+j-1 starting from nonterminal no. k if P[i,j,k] = true.

```
// initialize P with false and parse_tree with ""
for i=1 to n
   for j=1 to n-i+1
      for k=1 to 3 {
         P[i,j,k] = false;
         parse_tree[i,j,k] = "";
      }

// S -> L and L -> . lead to:
for i=1 to n {
   P[i,1,2] = true;
   parse_tree[i,1,2] = "4";
   P[i,1,1] = true;
   parse_tree[i,1,2] = "24";
```

```
}
for j=2 to n //length of the substring
   for i=1 to n-j+1 {//start of the substring
      if ( xi=="(" and xi+j-1==")" and P[i+1,j-2,3] )
         then {
            P[i,j,2] = true;
            parse_tree[i,j,2] = "3" + parse_tree[i+1,j-2,3];
            P[i,j,3] = true;
            parse_tree[i,j,3] = "5" + parse_tree[i+1,j-2,3];
            P[i,j,1] = true;
            parse_tree[i,j,1] = "23" + parse_tree[i+1,j-2,3];
         }
      for k=1 to j-1 { //partition of the substring
         if ( P[i,k,2] and P[i+k,j-k,1] )
            then {
               P[i,j,1] = true;
               parse_tree[i,j,1] = "1" + parse_tree[i,k,2] + parse_tree[i+k,j-k,1];
               P[i,j,3] = true;
               parse_tree[i,j,3] = "6" + parse_tree[i,k,2] + parse_tree[i+k,j-k,1];
            }
   }
```

if (P[1,n,1] == true) than parse_tree[1,n,1] is the parse_tree used to generate X.

Determine the probability of a sequence

Let ps and pd contain the probabilities for single and dinucleotides.
Let P[n,n,3] be an array of doubles: P[i,j,k] = the probability of obtaining xi...xi+j-1 starting from nonterminal no. k

```
// initialize P with 0
for i=1 to n
   for j=1 to n-i+1
      for k=1 to 3

         P[i,j,k] = 0;



// S -> L and L -> . lead to:

for i=1 to n {
   P[i,1,2] += (1-p2)·ps[xi];
   P[i,1,1] += (1-p1)·(1-p2)·ps[xi];
}
for j=2 to n //length of the substring
   for i=1 to n-j+1 {//start of the substring {
      P[i,j,2] += P[i+1,j-2,3] · p2 · pd[xi,xi+j-1];
      P[i,j,3] += P[i+1,j-2,3] · p3 · pd[xi,xi+j-1];
      P[i,j,1] += P[i+1,j-2,3] · (1-p1) · p2 · pd[xi,xi+j-1];
      for k=1 to j-1 { //partition of the substring
         P[i,j,1] += P[i,k,2] · P[i+k,j-k,1] · p1;
         P[i,j,3] += P[i,k,2] · P[i+k,j-k,1] · (1-p3);
   }
}
```

P[1,n,1] will contain the probability of sequence X.

Determine the structure with the highest probability for a sequence

The structure we are interested in is the one that maximizes

$$P(s \mid x) \cdot P(s \mid G)$$

Let parse_tree[n,n,3] be an array of strings: parse_tree[i,j,k] = the sequence of rules needed to generate the structure with highest probability for xi...xi+j-1 starting from nonterminal no. k.
Let P[n,n,3] be an array of doubles: P[i,j,k] = the probability of the structure generated by the parse_tree[i,j,k].

```
// initialize P with 0 and parse_tree with ""
for i=1 to n
   for j=1 to n-i+1
      for k=1 to 3 {

         P[i,j,k] = 0;
         parse_tree[i,j,k] = "";
      }

// S -> L and L -> . lead to:
for i=1 to n {
   P[i,1,2] = (1-p2) · ps[xi];
   parse_tree[i,1,2] = "4";

   P[i,1,1] = (1-p1) · (1-p2) · ps[xi];
   parse_tree[i,1,1] = "24";
}

for j=2 to n //length of the substring
   for i=1 to n-j+1 {//start of the substring {
      if ( P[i,j,2] > P[i+1,j-2,3] · p2 · pd[xi,xi+j-1])
         then {
            P[i,j,2] = P[i+1,j-2,3] · p2 · pd[xi,xi+j-1];
            parse_tree[i,j,2] = "3" + parse_tree[i+1,j-2,3];
         }
      if ( P[i,j,3] > P[i+1,j-2,3] · p3 · pd[xi,xi+j-1] )
         then {
            P[i,j,3] = P[i+1,j-2,3] · p3 · pd[xi,xi+j-1];
            parse_tree[i,j,3] = "5" + parse_tree[i+1,j-2,3];
         }
      if ( P[i,j,1] > P[i+1,j-2,3] · (1-p1) · p2 · pd[xi,xi+j-1] )
         then {
            P[i,j,1] += P[i+1,j-2,3] · (1-p1) · p2 · pd[xi,xi+j-1];
            parse_tree[i,j,1] = "23" + parse_tree[i+1,j-2,3];
         }
      for k=1 to j-1 { //partition of the substring
         if ( P[i,j,1] > P[i,k,2] · P[i+k,j-k,1] · p1 )
            then {
               P[i,j,1] += P[i,k,2] · P[i+k,j-k,1] · p1;
               parse_tree[i,j,1] = "1" + parse_tree[i,k,2] + parse_tree[i+k,j-k,1];
            }
         if ( P[i,j,3] > P[i,k,2] · P[i+k,j-k,1] · (1-p3) )
            then {
               P[i,j,3] += P[i,k,2] · P[i+k,j-k,1] · (1-p3);
               parse_tree[i,j,3] = "6" + parse_tree[i,k,2] + parse_tree[i+k,j-k,1];
            }
      }
   }
```

P[1,n,1] will contain the highest probability for sequence X and parse_tree[1,n,1] will contain the parse_tree needed to generate the structure that gives P[1,n,1] probability.

Appendix B: The RNAfold structures vs. predicted structures

old	miRNA 1	>cel-let-7 MI0000001 Caenorhabditis elegans let-7 stem-loop
	miRNA 2	>cel-mir-1 MI0000003 Caenorhabditis elegans miR-1 stem-loop
	miRNA 3	>cel-mir-2 MI0000004 Caenorhabditis elegans miR-2 stem-loop
new	miRNA 4	>hsa-mir-517a MI0003161 Homo sapiens miR-517a stem-loop
	miRNA 5	>hsa-mir-521-2 MI0003163 Homo sapiens miR-521-2 stem-loop
	miRNA 6	>hsa-mir-517b MI0003165 Homo sapiens miR-517b stem-loop
old	tRNA 1	>Anopheles_gambiae_chr3R.trna51-AlaAGC (12752402-12752330) Ala(AGC) 73 bp Sc:59.55
	tRNA 2	>Anopheles_gambiae_chr2R.trna116-ArgACG (11444246-11444174) Arg(ACG) 73 bp Sc:73.12
	tRNA 3	>Anopheles_gambiae_chr3R.trna17-ArgTCG (19628281-19628353) Arg(TCG) 73 bp Sc:58.44
new	tRNA 4	>Bos_taurus_chr22.trna155-GlyCCC (4638281-4638353) Gly(CCC) 73 bp Sc:35.66
	tRNA 5	>Bos_taurus_chr1.trna9631-GlyCCC (45528076-45528004) Gly(CCC) 73 bp Sc:35.67
	tRNA 6	>Bos_taurus_chr3.trna8812-GlyCCC (17236909-17236837) Gly(CCC) 73 bp Sc:35.68

The pictures are found in the attached file.