Using stochastic context-free grammars to predict RNA secondary structure: Project report

Sudhanshu Bharadwaj
Undergraduate, IISc
sudhanshub@iisc.ac.in

Note: All the code is linked at the GitHub repository

I Introduction

I-A. *RNA secondary structure estimation*

RNA, being a single-stranded molecule, can undergo base-pairing interactions. These interactions lead to various secondary structures which are important for its functionality. RNA nucleotides interact to form secondary structure motifs such as stems, loops, bulges, and pseudo-knots [2]. Given the tediousness of determining RNA secondary structures using crystallographic methods, there's a need for computer-based methods to predict RNA structures

These pairings often have a nested structure and complicated algorithms are needed to predict or model such structures. The most popular methodology to do this is by thermodynamic energy minimization [9] Alternatively, one can use probabilistic modeling approaches like Stochastic Context-Free grammars(SCFGs) [6], [8]. This is more advantageous because such probabilistic models can easily be extended to include other sources of statistical information to better enforce biological constraints.

RNA structure prediction can also benefit greatly from comparative sequence analysis. When RNA molecules are homologous, they usually retain a similar secondary structure with base pairing, and analyzing multiple RNA sequence alignments can reveal conserved base-pairing interactions through compensatory mutations. Therefore, using phylogenetic information can also improve RNA prediction [5].

I-B. *Precise formulation*

Our goal is to predict the positions of secondary structures in the RNA. But this problem can be simplified to finding out if a base is paired up, and to which base. If we know this, we can infer the correct base pairings and thus how the RNA will fold. One way to represent such secondary structure information is representing the information using a connect file format. Basically, we have a $5' \to 3'$ indexed list of base pairs, and the correspind indices of bases they are paired up with (0 if they are not paired)

Input: A list of train RNA secondary structure in .ct file format, and a list of test RNA sequences *Output*: Secondary structure prediction for the test sequences. i.e. for each base in the test sequence, the index of it's base pair(if any)

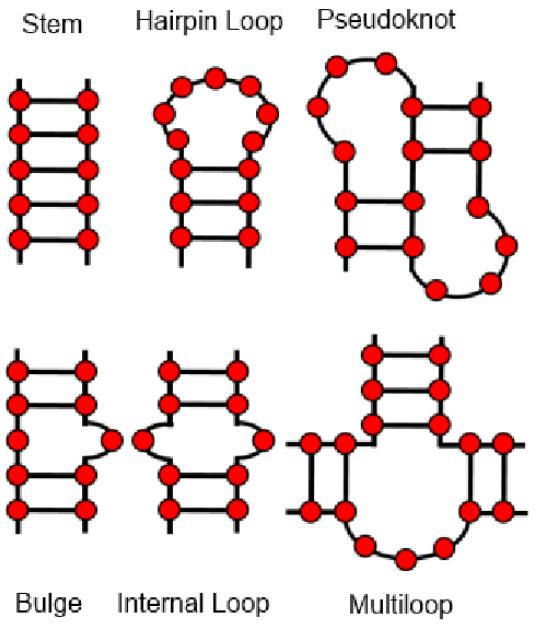


Fig. 1. Some secondary structural motifs [3]

II Theory and ALgorithms

II-A. *SCFGs*

Stochastic context-free grammars are one way to model such systems, leveraging regularities and base-pairing rules associated with the secondary structures. In this section, SCFGs shall briefly be introduced.

An SCFG can be represented as a tuple $(N, \Sigma, R, S, \mathcal{P})$, where:

- N is a set of nonterminal symbols
- Σ is a set of terminal symbols
- R is a set of production rules of the form $A \to \alpha$, where $A \in N$ and $\alpha \in (N \cup \Sigma)^*$
- S is the start symbol (a non terminal)
- $\mathcal{P}$ is a set of probability distributions over the production rules in R, such that $\sum_\alpha \mathcal{P}(A \to \alpha) = 1 \ \forall A \in N$

At each step, from the grammar, one has a current string of terminals and/or nonterminals; one chooses a nonterminal and transforms that nonterminal into a new substring using a given production rule. This process starts with the initial string S, iterates until one arrives at a string consisting solely of

terminal symbols. Such a derivation is called a parse. Each of the productions, and therefore the entire parse has a probability associated with it. In our case, terminals consist of nucleotide $\Sigma = \{A, U, C, G\}$.

A relatively simple yet powerful grammar is written below,

$$S \to LS \mid S \to L$$
$$L \to s \mid L \to dFd'$$
$$F \to LS \mid F \to dFd'$$

Here, s represents a base in a string and (d, d') represents paired bases in a stem. In the above grammar, The non-terminal S generates loops, F generates stems, and L determines if a loop position is a single base or the start of a new stem. Loops have a minimum length of two positions because F generates LS instead of just S. Two-position loops can be two bases, one base, and a new stem, or two new stems. In total, there are 36 free parameters in this grammar. An example of how a secondary structure can be derived from these rules is shown in Fig 2.

a) $S \to LS \to LLLLLLLS \to LLLLLLLL$
$\to ssLssssss \to ssdFdssssss$
$\to ssdddFdddssssss$
$\to ssdddLSdddssssss$
$\to ssdddLLLLdddssssss$
$\to ssdddssssdddssssss$

b)

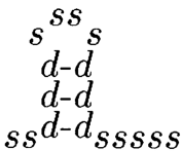


Fig. 2. Production of RNA structures by the grammar. (a) The rules being used, starting from S. (b) The corresponding structure [5]

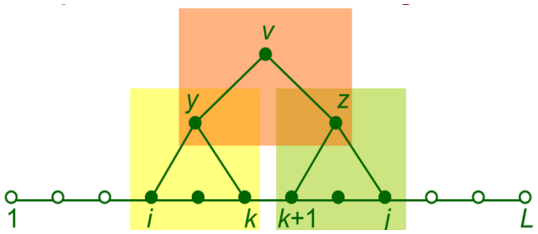
For a CFG these rules are genrerally in Chomsky-Normal-Form, i.e each rule is one of two types.

$$A \to BC \ (Transition)$$
$$A \to a \ (Emission)$$

where $A, B, C \in N$ and $a \in \Sigma$. Even though this simplifies notation and code, it greatly increases the parameters, and thus computation involved. Therefore, **all algorithms mentioned have been extended in the code for Non-Chomsky Forms**. Some algorithms have been discussed below

1) The Inside and Outside Algorithm

One interesting question would be to determine, Given a grammar G and sequence S, how likely is it that the sequence is generated using that grammar. Since a sequence can generally be parsed in multiple ways if we sum the probability across all possible parses, we shall have our answer.



This problem is perfectly suited for Dynamic programming, where we somehow calculate the probabilities at each transformation. Let α be a 3D matrix who's $(i, j, v)th$ element represents the probability that $P[i:j]$ has been generated by the non-terminal v. At each rule application point $v \to yz$, we can go through all the possibilities y, z, k, and sum probability of each resulting non-terminal. The algorithm is summarised below

Another algorithm that's a simple modification of this, is the outside algorithm, which calculates the "outside probability". It has a dynamic programming matrix $\beta(i, j, v)$ which denotes the probability of generating the sequence from S excluding the probability of generating $P[i:j]fromv$

Inside Algorithm:

- **Initialization** $(i = 1{:}L, v = 1{:}M)$
  $\alpha(i, i, v) = e_v(T_i)$
- **Recursion** $(i = L-1{:}1, j = i+1{:}L, v = 1{:}M)$

$$a(i,j,v) = \sum_{y=1}^{M}\sum_{z=1}^{M}\sum_{k=i}^{j-1} t_v(y,z) \times \alpha(i,k,y) \times \alpha(k+1,j,z)$$

- **Termination**
  $\mathbb{P}(T|\theta) = \alpha(1, L, 1)$
- **Time Complexity**
  $O(L^3M^3)$

Outside Algorithm:

- **Initialization** $(i = 1{:}L, v = 1{:}M)$
  $\beta(i, L, v) = (v == Start)$
- **Recursion** $(i = 1{:}L, j = L{:}i, v = 1{:}M)$

$$\beta(i,j,v) = \sum_{y=1}^{M}\sum_{z=1}^{M}\sum_{k=1}^{i-1} t_y(z,v) \times \alpha(k,i-1,z) \times \beta(k,j,y) \quad \text{when } y \to zv$$
$$+ \sum_{y=1}^{M}\sum_{z=1}^{M}\sum_{k=j+1}^{L} t_y(v,z) \times \alpha(j+1,k,z) \times \beta(i,k,y) \quad \text{when } y \to v,z$$

- **Termination**
  $\mathbb{P}(T|\theta) = \alpha(1, L, 1)$
- **Time Complexity**
  $O(L^3M^3)$

2) CYK algorithm and inferring secondary structure

This algorithm finds the MAP estimate for derivation used to generate the given text from the grammar. It gives us the rules that were applied at each step. Here we have two DP tables, one 4D and one 3D. $\gamma(i, j, v)$ stores the probability of the most

likely parse from v to generate $P[i:j]$, while $\tau(i,j,v)$ stores information about the next step in the derivation (which rule was applied, to generate which non-terminals, and how they were split). The initialization is similar to the inside algorithm. Here I won't describe the full algorithm, but an outline for the **non-CNF** version of our rule.

We determine the most likely applied rule at (i,j,V), by taking the maximum of the 3 rules: (i)

1) Transmission(like $S \rightarrow LS$): (i,j,V) splits into (i,k,Y) and $(k+1,j,Z)$
2) Replacement (like $S \rightarrow L$): (i,j,V) becomes (i,j,X)
3) Emission and Replacement (like $F \rightarrow dFd'$): (i,j,V) generates $P[i], P[j]$ and becomes $(i+1, j-1, X)$
4) Emission (like $L \rightarrow s$): (i,i,v) becomes $P[i]$

here i,j,k represent indices, s,d,d' terminals and V,X,Y,Z represent non-terminals.

Here is where the ingenuity of the rule set comes in. If an emission rule(iv), the base generated is not paired. However, if an Emission and replacement rule(iii) is used, then a base pair is formed. Therefore from this algorithm, one can **directly infer all stem and loop regions**.

3) Training a SCFG

Perhaps the most important question is to infer the parameters of SCFG. (i.e the probabilities associated with each rule) This is done using the Inside Outside Algorithm, which is an Expectation Maximization algorithm

Give a Pattern, the goal is to maximize the likelihood $\alpha[1,L,S]$ from the inside algorithm. This is done by modifying the probabilities of the rules to make it more likely to generate the text. At each step we calculate the number of times(probabilistically speaking) a given rule was used to generate the text (conditioned on the old probabilities). Based on this we calculate the new probability of a given rule.

$$\mathcal{P}_{new}(v \rightarrow w) = \frac{c(v \rightarrow w|\mathcal{P}_{old})}{\sum_x c(v \rightarrow x|\mathcal{P}_{old})}$$

here v is a non-terminal and x, w are strings (a combination of non-terminals and terminals).

This can easily be done using α and β from the inside and outside algorithms. At all positions where the rule can be applied, we multiply the outside probability, the inside probability of the derived characters, and the probability of the rule. This estimates the total probability of all the derivations using the above rule. An example calculation has been shown for the case of only Emissions and Transmissions.

$$\hat{e}_v(A) = \frac{c(v \rightarrow A)}{c(v)} = \frac{\sum_{i|x_i=A} \beta(i,i,v) e_v(A)}{\sum_{i=1}^{L}\sum_{j=i}^{L} \beta(i,j,v)\alpha(i,j,v)}$$

cases where v used to generate A

cases where v used to generate any subsequence

$$\hat{t}_v(y,z) = \frac{c(v \rightarrow yz)}{c(v)} = \frac{\sum_{i=1}^{L-1}\sum_{j=i+1}^{L}\sum_{k=i}^{j-1} t_v(y,z)\,\beta(i,j,v)\,\alpha(i,k,y)\,\alpha(k+1,j,z)}{\sum_{i=1}^{L}\sum_{j=i}^{L}\beta(i,j,v)\,\alpha(i,j,v)}$$

Even though this algorithm has very high complexity (cubic in a number of rules and string length), for our simple grammar we just need a few iterations of EM to converge to the desired algorithm. We shall see how we use the statistical properties of our data to further reduce the number of free parameters

II-B. *Simple statistical trick*

As we've seen above, in the grammar that we are using, the rules:

$$L \rightarrow s \ (unpaired\ base \ : loop)$$
$$L \rightarrow dFd' \ (paired\ basepairs \ : stem)$$
$$F \rightarrow dFd' \ (paired\ basepairs \ : stem)$$

represent actually represent 4/16 rules. But using properties of real data, we can reduce this problem to only calculating the overall probabilities. Using previously labelled data, calculate tables $T_1(4 \times 1)$ and $T_2(4 \times 4)$. Each row/ column here represents one of $\Sigma = A, U, C, G$.

- In T_1 we estimate the percentage of each base pair in the unpaired regions
  $\therefore\ \mathcal{P}(L \rightarrow s) = p_1 \times T_1[s]$
- In T_2 we estimate the percentage of base-pair in the paired regions.
  $\therefore\ \mathcal{P}(L \rightarrow dFd') = p_2 \times T_2[d,d']$
  $and\ \mathcal{P}(L \rightarrow dFd') = p_3 \times T_2[d,d']$

where p_1, p_2, p_3 are the probabilities of the overall rules

This way, we have reduced the number of free parameters in our model down to 3, which makes EM much faster. If our inside algorithm is fast enough, one could also use minimization methods like gradient descent in conjunction.

II-C. *Mutation Model and using phylogeny*

The last improvement in this algorithm is using phylogenetically related sequences to predict the structure better. The key idea is that, even though some base pairs RNA might be different in related secondary structures, the secondary structures ares still conserved. Using a GTR model to model mutations over time, one can use this property to an advantage. Using a training corpus of related , one can calculate the probability of different mutation sets in the stem and loop regions. (In pairs for stem regions). These would be different because the base-pairing in stem regions cause more transition type mutations over transversions.

The production rules can now be modified to produce n base-pairs where n is the number of elements in the phylogeny. Fore example, if we had two sequences, the rule $L \rightarrow \{A,U,C,G\}$ would now be $L \rightarrow \{A,U,C,G\}^2$ (16 rules from 4). These free parameters don't add parameters to the Grammar because as before we just multiply the replacement rates from data with the original rules to generate rules for n-sequences . Below is a derivation of outlaying how to use the mutation model in the grammar.

Let $C_1, C_2 ... C_l$ represent the columns of an alignment, where l is the length of the sequence, and each of these elements is n vectors long, where n is the number of elements in

the phylogeny. Let D be the entire data $(C_1, C_2...C_l)$, T be the allignemet tree and σ be different secondary structures(loops and strands). The likelihood of the given data can be summed across different structures:

$$P(D|T,M) = \sum_\sigma P(D,\sigma|T,M)$$
$$= \sum_\sigma P(D|\sigma|T,M)P(\sigma|T,M)$$
$$= \sum_\sigma P(D|\sigma,T,M)P(\sigma|M)$$

because we have assumed the secondary structures to be independent of the tree. Now, using the assumption that mutation on each base is independent,

$$P(D|\sigma,T,M) = P(C_1, C_2..C_n|\sigma,T,M)$$
$$= \prod_s P(C_s|\sigma,T,M) \prod_{d,d^c} P(C_d C_{d_c}|\sigma,T,M)$$

The first term is over single bases, and the second is over pairs.The tree can also be found using maximum likelihood as $T^{ML} = argmax_T P(D|T,M)$

However, due to the absence of data-sets with many related sequences, alignment has not been used in the implementation

III Results

All results presented, unless stated otherwise, were produced by code written entirely by me in python using standard libraries(numpy, pandas,matplotlib) and numba to **JIT-compile** code to make it faster.

Data was taken from [1]. The RFA and SRP databases were selected. First, all sequences larger than 300 sequences were removed(to avoid float underflow). This was then split randomly into a 50%,25%, 25% train, test, and validation split(This corresponded to 270,135 and 135 sequences respectively). IN the .ct files, all unpaired regions were considered loops and paired regions stems. Ambiguous/ unidentified nucleotides were randomly assigned one of A,U,C,G.

III-A. *RNA statistics*

Below are the ratio of single nucleotides in the loop regions

Nucelotide	A	C	G	U
Frequency	0.33	0.21	0.22	0.24

For nucleotide pairs, the probabilities $T_2(i,j)$ and $T_2(j,i)$ are averaged. Below are the pair probabilities for stem regions.

Nucleotide	A	C	G	U
A	0.001	0.002	0.22	0.153
C	0.002	0.000	0.263	0.001
G	0.001	0.263	0.001	0.078
U	0.153	0.001	0.078	0.001

These frequencies were incorporated into the corresponding production rules

III-B. *Grammar parameters*

The training data is used to learn the parameters using the Inside-Outside algorithm. The EM algorithm was run for a maximum of 20 steps or until the likelihoods converged (in almost all runs they converged within 2 iterations).

The parameters were initialized randomly and this procedure was repeated 10 times, to avoid local minimas in the negative log likelihood space and over-fitting. In each of these runs, the MAP likelihood (using the CYK algorithm) of the validation set was chosen. The parameter set with the best validation score was finally selected

Below are the converged probabilities of each of the Macro Rules.

$$S \to LS \ (0.841) \mid S \to L \ (0.159)$$
$$L \to s \ \ (0.586) \mid L \to dFd' \ (0.414)$$
$$F \to LS \ (0.127) \mid F \to dFd' \ (0.873)$$

III-C. *Accuracy of results*

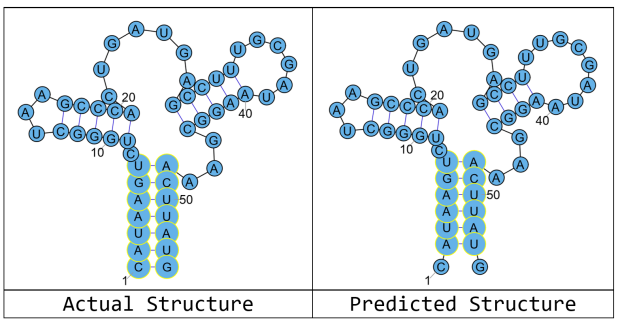
Using the above parameters and the CYK algorithm, secondary structures were predicted for the sequences. Using the following metrics, the TP, TN, FP and FN are calculated. The sensitivity and specificity, Positive predicitve value and negative value of the predictions is reported below:

- **Paired**: Here, just check if the prediction correctly identifies whether or not a given base is paired up or not
- **Precise**: Here, we not only check if the prediction identifies a base pair correctly or not, but also if it identifies the exact base it pairs up with. This is the harshest testing method.
- **Slide**: The same as precise, but the predicted base pair can be off by one base pair. (This won't effect the secondary structure much)

Method	Sens	Speci	PPV	NPV	% correct
Paired	71%	68%	74%	64%	70%
Slide	52%	69%	68%	51%	59%
Precise	51%	68%	68%	51%	58%

The predictions are pretty good as we can see above. It matches the values generated by a review paper using the same grammar [4]. According to the main paper [5], the result will improve a lot after using alignment of 2-4 sequences, but I haven't verified that.

I have visualised one of the predictions using a visualiser [7] (by writing out a .ct text file). The structures are almost the same, except our prediction has missed base pairs at the end of 2 stems. Perhaps, this could be fixed using some post-processing.



This brings us to the end of the report. It was really interesting to see a formal grammar concept working very well in such a bioinformatic setting.

References

[1] M. Andronescu, V. Bereg, H. H. Hoos, and A. Condon. Rna strand: The rna secondary structure and statistical analysis database - bmc bioinformatics, Aug 2008.

[2] P. Baldi and S. Brunak. *Bioinformatics: The Machine Learning Approach*. 01 2001.

[3] N. Chheda and M. Gupta. Rna as a permutation. 03 2014.

[4] R. D. Dowell and S. R. Eddy. *BMC Bioinformatics*, 5(1):71, 2004.

[5] B. Knudsen and J. Hein. RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. *Bioinformatics*, 15(6):446–454, June 1999.

[6] B. Knudsen and J. Hein. Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Research*, 31(13):3423–3428, 07 2003.

[7] J. S. Lu, E. Bindewald, W. K. Kasprzak, and B. A. Shapiro. RiboSketch: versatile visualization of multi-stranded RNA and DNA secondary structure. *Bioinformatics*, 34(24):4297–4299, June 2018.

[8] Y. Sakakibara, M. Brown, R. Hughey, I. S. Mian, K. Sjölander, R. C. Underwood, and D. Haussler. Stochastic context-free grammers for tRNA modeling. *Nucleic Acids Research*, 22(23):5112–5120, 11 1994.

[9] M. Zuker. [20] computer prediction of rna structure. In *Methods in enzymology*, volume 180, pages 262–288. Elsevier, 1989.