Stochastic Context-Free Grammars for RNA Analysis

Heitor Baldo*
Algorithms in Bioinformatics (IBI5037)

Spring, 2019

Abstract

In this review, we discuss stochastic context-free grammars (SCFGs) for analyzing and predicting RNA secondary structures. We briefly present the classical theory of SCFGs, including some examples and algorithms. Subsequently, we discuss the problem of grammar ambiguity, grammar design methods, and, finally, their applications to RNA secondary structure prediction.

Contents

*NUSP: 11571801 / hbaldo@usp.br

1 Introduction

Predicting RNA secondary structure is a long-standing challenge in the field of bioinformatics. The problem is to derive a two-dimensional structure from the primary sequence (nucleotide sequence) of RNA. The aim is to predict the position of hydrogen bonds in an RNA molecule based only on its primary sequence. These predictions can be used to better understand the functioning of cells, the characteristics of gene expression, and the mechanisms involved in protein production.

Several computational methods have been developed to solve this problem, but it has not yet been possible to develop an optimal method. The problem becomes even more computationally difficult when allowing the occurrence of a certain type of substructure, namely *pseudoknots*, which occur in real RNA structures and are therefore relevant in computational modeling.

The main focus of this review consists of the analysis of the articles [1, 2, 6], focusing on *context-free stochastic grammars* (SCFGs) and how they can be used to provide a probabilistic approach to predicting RNA secondary structures. We will briefly discuss the theory behind SCFGs, including the main algorithms for SCFGs, grammar ambiguities, methods for obtaining these grammars, and their applications in the analysis of RNA sequences.

2 Stochastic Context-Free Grammars

Before we formally define a SCFG, let's define a *context-free grammar* (CFG).

2.1 Context-Free Grammars

Definition 2.1. A CFG G is a 4-tuple (V, T, P, S) where:
(i) V is a finite set of non-terminals ("states");
(ii) T is a finite set of terminals;
(iii) P is a finite set of production rules;
(iv) S is the initial non-terminal ($S \in V$).

Example 2.2. The following CFG generates RNA structures: $V = \{s\}$, $T = \{A, C, U, G\}$ and

$$P = \begin{cases} s \to AsU \mid UsA \mid CsG \mid GsC \\ s \to As \mid Cs \mid Gs \mid Us \\ s \to sA \mid sC \mid sG \mid sU \\ s \to ss \\ s \to \epsilon \end{cases}$$

Example 2.3. Another simple CFG that generates RNA structures is: $V = \{S_0, S_1, S_2, ..., S_{13}\}$, $T = \{A, C, U, G\}$, with a set of production rules P as shown below:

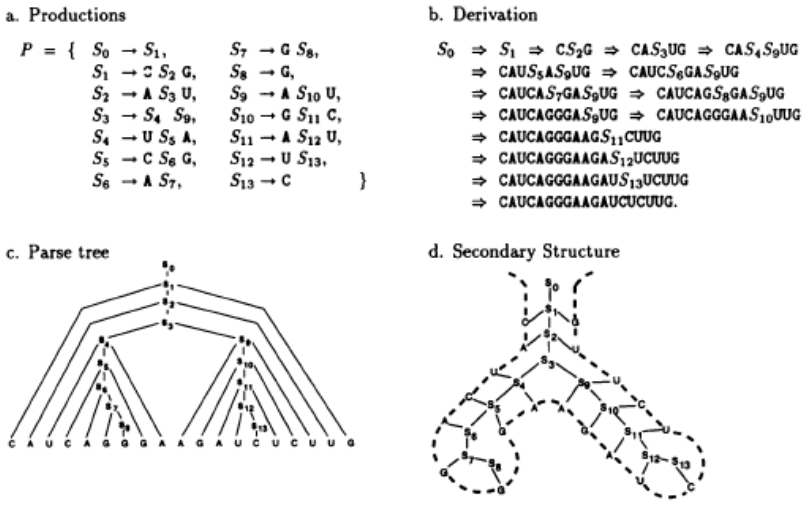

Figure 1. A CFG that can produce a set of RNA molecules. A *derivation* of CFG (b) has a graphical representation known as *derivation tree* (c). The secondary structure obtained from derivation (b) (without pseudoknots) is shown in (d) (figure reproduced from [6]).

Observation 2.4. CFGs are not useful for structure prediction because, given an RNA sequence, there may be a large number of valid derivation trees, each corresponding to a possible RNA secondary structure (they are ambiguous). To try to overcome this type of problem, we use SCFGs.

2.2 Stochastic Context-Free Grammars

Let us now formally define a stochastic context-free grammar.

Definition 2.5. An SCFG is a CFG, $G = (V, T, P, S)$, such that the set of production rules P has a probability distribution. That is, an SCFG specifies a probability for each production in the grammar.

We denote the total set of probabilities (the *parameters* of the model) by θ. We can thus denote a SCFG by (G, θ).

Example 2.6. A simple SCFG is defined as follows: $V = \{S, w_1, w_2, w_3\}$, $T = \{A, C, U, G\}$,

$$P = \begin{cases} S \to Aw_1U^{0.25} \mid Cw_1G^{0.25} \mid Gw_1C^{0.25} \mid Uw_1A^{0.25} \\ w_1 \to Aw_2U^{0.25} \mid Cw_2G^{0.25} \mid Gw_2C^{0.25} \mid Uw_2A^{0.25} \\ w_2 \to Aw_3U^{0.2} \mid Cw_3G^{0.3} \mid Gw_3C^{0.3} \mid Uw_3A^{0.2} \\ w_3 \to GAAA^{0.2} \mid GCAA^{0.8} \end{cases}$$

The superscript numbers represent the probabilities associated with each production. Note that the sum of the probabilities associated with the productions are always equal to 1.

Observation 2.7. By definition, an SCFG describes a joint probability distribution $P(x, \pi|G, \theta)$, given a sequence x, over all possible derivation trees π, that is, $P(x, \pi|G, \theta)$ is the product of all probabilities of the production rules used in a derivation tree π for a sequence x.

2.3 Chomsky Normal Form

To develop algorithms to analyze sequences using grammar models, it is convenient to restrict them to a certain *normal form* (which has only a few types of productions). The most commonly used normal form is *Chomsky normal form* (CNF), formally defined below.

Definition 2.8. We say that a CFG is in CNF if all its production rules are of the form $A \to YZ$ or $A \to a$ or $S \to \epsilon$, where A, S, Y, Z are nonterminal, a is terminal, and ϵ is the empty string.

Any CFG can be put into this normal form, that is, every CFG is equivalent to a grammar in the CNF, and, probably, every SCFG can be converted to a set of rules of this type.

Observation 2.9. Despite CNFs being useful, a CNF grammar cannot introduce paired nucleotides from a single production and, therefore, does not capture the RNA structure directly. Therefore, in applications, it is convenient to avoid this conversion. However, a new normal form was constructed in [1], such that it could capture the fundamental characteristics of RNA secondary structure: branching, unpaired bases and paired bases. This normal form is defined as follows.

Definition 2.10. For non-terminals T, U, V, the *double emission normal form*[1] has only rules of the form $T \to UV$ or $T \to .$ or $T \to (U)$, where $(,)$ represents base pairs and $.$ represents null emission.

This normal form will be used in the evolutionary approach to searching for grammars, as we will see later.

2.4 Algorithms for SCFGs

In general, when we work with SCFGs, we use algorithms for three main purposes:

1) **Optimal derivation tree:** A dynamic programming algorithm for SCFGs that can determine the most likely derivation tree for a sequence x with time complexity proportional to L^3, where L is the size of x, is the *CYK* (Cocke-Younger-Kasami) algorithm.

2) **Sequence Probability:** The *Inside algorithm* is a dynamic programming algorithm (analogous to the *forward* algorithm for HMMs) that is used to obtain the total derivation probability of a sequence x, given the model (G, θ), summing over all possible derivation trees.

3) **Parameter Estimation:** One of the most commonly used algorithms for estimating the parameters of a SCFG (i.e., the production probabilities), from a set of training sequences, is the *Inside-Outside algorithm* (IO). Just like the *forward-backward* algorithm for HMMs, this algorithm is an Expectation Maximization (EM) method to obtain the highest probability of the SCFG parameters.

[1] So named because paired bases are emitted simultaneously.

2.4.1 The CYK Algorithm

The CYK algorithm calculates the most likely derivation tree (and therefore the secondary structure) for a sequence x (of length L), given a SCFG (G, θ). For a SCFG, the CYK algorithm is characterized by the following steps:

Initialization: $\gamma(i, i-1) = log\ p(S \to \epsilon)$

Iteration:

$$\gamma(i,j) = max \begin{cases} \gamma(i+1, j-1) + log\ p(S \to x_i S x_j) \\ \gamma(i+1, j) + log\ p(S \to x_i S) \\ \gamma(i, j-1) + log\ p(S \to S x_j) \\ max_{i<k<j}\{\gamma(i,k) + \gamma(k+1,j) + log\ p(S \to SS)\} \end{cases}$$

Termination: $\gamma(1, L) = log\ P(x, \hat{\pi}|G, \theta)$

Thus, the CYK algorithm finds an optimal derivation tree $\hat{\pi}$ for the sequence x by doing the traceback:

$$\hat{\pi} = arg\ max_{\pi}\ P(x, \pi|G, \theta).$$

2.4.2 Parameter Estimation

We can learn the parameters of an SCFG from training sequences using an EM approach called *Inside-Outside* (IO) algorithm [5]. This algorithm can be seen as a generalization for SCFGs of the *forward-backward algorithm* for parameter estimation in HMMs.

However, in [6] an alternative algorithm, called *Tree-Grammar EM training algorithm*, was used to estimate the parameters of a SCFG from non-aligned training RNA sequences. After designing a suitable initial grammar, training sequences are employed only to refine estimates of the grammar's production probabilities.

In [2], the parameters of each SCFG were estimated from frequencies observed in annotated secondary structures. The training set was composed of large and small rRNA subunits obtained from *European Ribosomal RNA Database*[2].

In [1], both the CYK algorithm[3] as well as the Inside-Outside algorithm, for training the grammars found by the evolutionary approach.

2.5 Grammar Ambiguity

Definition 2.11. A grammar is said to be *ambiguous* when there is more than one possible derivation tree for some sequence. Furthermore, when multiple derivation trees describe the same secondary structure, we call the grammar *structurally ambiguous*.

[2] http:/ /bioinformatics.psb.ugent.be/webtools/rRNA/

[3] For the CYK algorithm, in the case of ambiguous grammars, it is not possible to know which derivation produced the known structure; therefore, probabilities cannot be obtained. Consequently, we train these grammars by randomly selecting a derivation.

Example 2.12. The following grammar is ambiguous: $V = \{S\}$, $T = \{a, c, g\}$,

$$P = \begin{cases} 1.\ S \to gSc \\ 2.\ S \to gS \\ 3.\ S \to aa \end{cases}$$

With this grammar we have three possible derivation trees that generate the string $GGGAACC$, namely, by applying the rules $2, 1, 1, 3$ and $1, 2, 1, 3$ and $1, 1, 2, 3$.

Observation 2.13. If a structure A has a derivation with probability 0.3 and a structure B has two derivations with probability 0.25 each, the CYK algorithm will choose structure A, despite structure B being more likely . In other words, ambiguity can reduce the quality of predictions made with the CYK algorithm.

Grammar ambiguity is a problem in RNA sequence analysis. For example, if a grammar is structurally ambiguous, we cannot equate the probability of a derivation tree with the probability of its structure. Thus, an optimal structure cannot be efficiently obtained by the CYK algorithm, forcing us either to use structurally unambiguous (unambiguous) grammars, or we will have to assume that it is a valid approximation to assume that an optimal derivation tree provides us with the optimal structure.

Ambiguity is one of the reasons why we want to develop methods of searching for grammars that are good enough, as we will see below.

3 Finding Grammars

Firstly, it would be natural to search for grammars using the brute force method. In this case, we would have to construct and evaluate the effectiveness of relatively small grammars. This was done in [2], and we will investigate the results obtained below.

3.1 Brute Force

In [2] nine hand-built grammars were analyzed. Two of them proved to be ambiguous (namely, $G1$ and $G2$), and the other seven proved to be unambiguous.[4] Next, we will analyze the grammars $G3, G4, G5$ and $G6$.

Example 3.1. For terminals $a, \hat{a}$ and non-terminals S, L, R, F, we have four unambiguous grammars:

$$G3 : \begin{cases} S \to aS\hat{a} \mid aL \mid Ra \mid LS \\ L \to aS\hat{a} \mid aL \\ R \to RA \mid \epsilon \end{cases}$$

[4]For simplicity, the four grammars G2, G7, G8 and G9, were not considered in this text as they involve productions of the type $p^{b\hat{b}} \to aP^{a\hat{a}}\hat{a}$, in which the probability of issuing a new pair a, $hata$ depends on the previous base pair $b, \hat{b}$ ($a, \hat{a}, b, \hat{b}$ terminals, and $P^{a\hat{a}}, P^{b\hat{b}}$ non-terminals). The notation $P^{a\hat{a}}$ is used to include base pair stacking parameters (stacking grammar).

$$G4: \begin{cases} S \to aS \mid T \mid \epsilon \\ T \to Ta \mid aS\hat{a} \mid TaS\hat{a} \end{cases}$$

$$G5: S \to aS \mid aS\hat{a}S \mid \epsilon$$

$$G6: \begin{cases} S \to LS \mid L \\ L \to aF\hat{a} \mid a \\ F \to aF\hat{a} \mid LS \end{cases}$$

Each of the four grammars has been conjectured to be unambiguous by inspection. Each also passed the empirical ambiguity test described in [2]. See the same article for examples of derivation trees for these grammars.

Remark 3.2. Each of these grammars imposes slightly different restrictions on possible structures. $G3$ imposes a minimum *hairpin loop* length of one nucleotide, $G6$ has a minimum of two, and $G4$ and $G5$ do not impose minimum *hairpin loop* lengths. Furthermore, $G6$ cannot emit an empty string, ϵ, while $G3$, $G4$ and $G5$ can.

The accuracy of each grammar for secondary structure prediction was measured by calculating sensitivity and precision (PPV) on a *benchmark* set of 403 reliable secondary structures derived from comparative analysis. From this, it was found that the $G5$ grammar had the worst prediction performance; $G3$ had an average performance, while $G6$ had the best performance among the four grammars.

Furthermore, the performances of these grammars were compared with those of some algorithms that use energy minimization to predict secondary structure, namely *mfold*, Vienna RNA, PKNOTS and RNAstructure, for the same set of *benchmark* . The result was that all four grammars performed worse than these algorithms.

3.2 Evolutionary Approach

As we mentioned before, in [2] nine hand-built grammars were analyzed, and there was little motivation for building their production rules. Therefore, a computational search in a larger space of grammars can find better grammars. However, in the exhaustive search for small grammars, computational problems arise almost immediately. In view of this, we must look for efficient computational methods. An example is the article [1], in which an evolutionary approach was used to search for new grammars.

Evolutionary Algorithm

We must place our evolutionary algorithm in *double emission normal form* (presented in section 2.3), to have an efficient search. For this normal form, for non-terminal m, there are $2^{m^3+m^2+m}$ grammars (m^3 production rules of type $T \to UV$, m^2 of type $T \to (U)$ and m of type $T \to .$).[5] The way the evolutionary algorithm searches the space is determined by the choice of

[5] In view of this, brute force search is computationally viable for grammars with only two nonterminals, in addition to the rule $T \to .$ ($2^{14} = 16,384$ grammars), but not for three nonterminals ($2^{39} = 549,755,813,888$ grammars).

initial population and the methods of *mutation*, *reproduction* and *selection*. Next, we will briefly describe the evolutionary method and the results obtained in [1].

Initial Population: We start with an initial population of small grammars and use mutation and reproduction rules to increase the number of non-terminals and production rules. The initial population was composed of sixteen grammars of the form:

$$S \to \left\{ \begin{array}{c|c|c|c|c|} SS & SB & BS & BB & (S) \\ - & - & - & - & \end{array} \right. .$$
$$B \to .$$

Mutation: Mutations constitute the majority of movement in the search space, so they are particularly important. They give the grammar new characteristics, allowing greater structural freedom and add production rules that can be used immediately or that can remain inactive. For non-terminal $V_i \in V$, and corresponding production rules P_{V_i}, the allowed stochastic mutations were:

(i) The initial non-terminal (and the corresponding production rules) changes;

(ii) A production rule is added or deleted;

(iii) A new non-terminal V' is added along with two new rules that guarantee that V' is reachable and that $P_{V'}$ is not empty;

(iv) A non-terminal is created with identical rules to an existing one;

(v) A production rule of the form $V_i \to V_j V_k$ is changed to $V_i \to V_j V_l$, or $V_i \to V_l V_k$, or $V_i \to V_l V_p$, or the production rule of the form $V_i \to (V_j)$ is changed to $V_i \to (V_k)$.

Reproduction: The reproduction model produces a grammar that can produce all the derivations of the grammars from which it originated (parent grammars). The grammar G formed from "reproduction" of the grammars G_1 and G_2 (with initial symbols S_1 and S_2) has initial symbol S, non-terminals $V_1, ..., V_n$, $W_1, ..., W_n, B$, terminals $(,), .$ and rules of productions:

(i) $P_S = P_{S_1} \cup P_{S_2}$;

(ii) For V_i: P_{V_i} are all occurrences in which S_1 is replaced by S;

(iii) For W_i: P_{W_i} are all occurrences where S_2 is replaced by S.

Selection: We increase the population in each generation by introducing several newly mutated or created grammars, and then reduce this number to a fixed-size population by stochastic elimination.

In the experiments, more than 300,000 grammars were researched. Several strong grammars were found using both CYK and IO algorithms. The analyzed grammars were denoted by $GG1, GG2, GG3, GG4$, $GG5$, $GG6$. Furthermore, the grammar $KH99'$, which is KH99 (Knudsen-Hein, 1999 [4]) in the normal double-emission form, was also included.

These grammars had sensitivity and PPV values of approximately $0.4-0.6$. Yet, as expected, grammars with similar structures had similar predictions. The difference in performance between KH99′ and KH99 was small, and this confirms that the representation of KH99 as KH99′ is good.

The $GG1-GG6$ grammars were compared with methods that use thermodynamic principles, such as UNAfold and RNAfold, and the results showed that these methods continue to perform better than methods that use SCFGs.

Since these grammars are designed to predict RNA secondary structure using the same training set, we would expect some similarity in the predictions. This was confirmed by the results, suggesting that new grammars produce different types of structures, thus being good representations of RNA secondary structures.

4 SCFGs in the prediction of RNA secondary structures

Most algorithms for secondary structure prediction use energy minimization. However, probabilistic approaches using SCFGs can be used. The use of SCFGs in predicting RNA secondary structures was based on the success of HMMs for modeling proteins and genes. In fact, SCFGs are a generalization of HMMs.

The secondary structure can be predicted using two methods:

1) **CYK Algorithm**: The CYK algorithm determines, through dynamic programming, the likelihood of the most likely derivation, then traceback is performed to find the most likely structure.

2) **Posterior Decoding**: One can employ a *posterior decoding* method using base-paired probability matrices. These probability matrices for a SCFG are obtained using the IO algorithm. Hence, the secondary structure with the maximum expected number of correct positions can be calculated through dynamic programming.

There are several approaches that use SCFG to predict RNA secondary structures, such as the Pfold and RNAalifold algorithms. Pfold [3] is based on the KH99 algorithm. This algorithm uses an SCFG to produce an initial probability distribution of RNA structure[6].

Obviously, RNA secondary structure prediction is only useful for RNA molecules that are structured, such as tRNA molecules, as opposed to mRNA, which are relatively unstructured. In [6], SFCGs were used for prediction of tRNA secondary structures.

One of the weaknesses of SCFGs for this type of prediction is that they cannot predict pseudoknots.

5 Final Considerations

In this brief work, we studied how the design of SCFGs impacts the prediction of RNA secondary structures and how the structural ambiguity of grammars can be harmful in the prediction of these structures. Also, we studied methods for searching for good enough grammars that, in a certain way, may circumvent these problems.

The brute force search method proved to be computationally inefficient for large grammars and that is why we considered the evolutionary method, which proved to be relatively efficient in the search for stronger grammars. Nevertheless, the accuracy of secondary structure prediction by SCFGs was shown to be slightly lower than that of methods that use thermodynamic principles, such as energy minimization.

[6]Namely, Pfold uses the $G6$ grammar from the previous section.

References

[1] Anderson, J., Tataru, P., Staines, J. et al., *Evolving stochastic context-free grammars for RNA secondary structure prediction.* BMC Bioinformatics (2012) 13, 78.

[2] Dowell, R. D. and Eddy, S. R. *Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction.* BMC Bioinformatics (2004) 5:71.

[3] Knudsen, B. and Hein, J. *Pfold: RNA secondary structure prediction using stochastic context–free grammars.* Nuc Acid Res (2003) 31: 3423–3428.

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

[5] Lari, K. and Young, S., *The estimation of stochastic context–free grammars using the inside–outside algorithm.* Comput Speech Language (1990) 4:35–56.

[6] Sakakibara, Y., Brown M., Hughey R., Mian I. S., Sjölander K., Underwood R. C., Haussler D., *Stochastic context-free grammars for tRNA modeling.* Nucleic Acids Res. (1994) 22(23):5112-20.