

A Genetic Algorithm for Grammars

James Anderson and Joe Staines

July 1, 2010

Background and Motivation

Ribonucleic acid (RNA) secondary structure prediction is an important problem in molecular biology; almost as soon as RNA started to be sequenced, methods have been established to determine the structure from the sequence of nucleotides. From the secondary structure one can start to find the function of a given RNA molecule, which is of course important in understanding how the cell works, characteristics of gene expression and the mechanisms involved in protein production.

One of the main methods for folding aligned sequences of RNA was based on the success of Hidden Markov Models (HMMs) in protein and gene modeling [10]. Scientists wanted to apply the HMMs to RNA secondary structure, but in an HMM, all positions are independent [14], something which is certainly could not be extended to a model for RNA. Thus SCFGs, as generalisations of HMMs, were used to model RNA secondary structure, initially by Sakakibara et al. (1994) [16], and then by many others, for example [11] and [15]. Notably Knudsen and Hein (1999) & (2003) created the Pfold algorithm [8], [9] which was shown to be effective in comparison with other SCFGs [4].

While Knudsen and Hein's grammar was effective, it seems to have been chosen relatively arbitrarily, in that there is little discussion about what motivated the choice of production rules. This problem was addressed by Dowell and Eddy (2004) [4] where nine different 'lightweight' (i.e. a small number of parameters) SCFGs were evaluated for single sequence structure on a benchmark set of RNA secondary structures (see Figure 1). Lightweight grammars were considered here for their computational advantages and simplicity, since if one would like to extend SCFGs to model multiple sequences, a combinatorical explosion of parameters can occur if this is not kept small.

G1: $S \rightarrow aSa \mid aS \mid Sa \mid SS \mid \epsilon$	G3: $S \rightarrow aS\hat{a} \mid aL \mid Ra \mid LS$	G7: $S \rightarrow aP^{aa}\hat{a} \mid aL \mid Ra \mid LS$	G6 ^S : $S \rightarrow LS \mid L$
G2: $S \rightarrow aP^{aa}a \mid aS \mid Sa \mid SS \mid \epsilon$	$L \rightarrow aS\hat{a} \mid aL$	$L \rightarrow aP^{aa}\hat{a} \mid aL$	$L \rightarrow aF\hat{a} \mid a$
$p^{bb} \rightarrow aP^{aa}a \mid S$	$R \rightarrow Ra \mid \epsilon$	$R \rightarrow Ra \mid \epsilon$	$F^{bb} \rightarrow aF^{aa}\hat{a} \mid LS$
	G4: $S \rightarrow aS \mid T \mid \epsilon$	$p^{bb} \rightarrow aP^{aa}\hat{a} \mid aN\hat{a}$	
	$T \rightarrow Ta \mid aS\hat{a} \mid TaS\hat{a}$	$N \rightarrow aL \mid Ra \mid LS$	
	G5: $S \rightarrow aS \mid aS\hat{a}S \mid \epsilon$	G8: $S \rightarrow aS \mid T \mid \epsilon$	
		$T \rightarrow Ta \mid aP^{aa}\hat{a} \mid TaP^{aa}\hat{a}$	
	G6: $S \rightarrow LS \mid L$	$p^{bb} \rightarrow aP^{aa}\hat{a} \mid aN\hat{a}$	
	$L \rightarrow aF\hat{a} \mid a$	$N \rightarrow aS \mid Ta \mid TaP^{aa}\hat{a}$	
	$F \rightarrow aF\hat{a} \mid LS$		

Figure 1: The grammars used by [4]. The left column is two structurally ambiguous grammars (G1 simple and G2 stacking), the middle column four unambiguous simple grammars, and the right two columns three unambiguous stacking grammars. Note the presence of the Knudsen-Hein Pfold grammar G6, which is extended to a stacking grammar in G6^S. Probabilities for each were estimated from training data.

Of course, there are many more grammars than these tested by Dowell and Eddy (2004), and it is quite possible that one of these is significantly stronger than any one seen at this point. It is not possible to test all possible grammars to see which is the best, and even with putting severe limitations on the form of our grammar it is unlikely that one would be able to search heuristically.

Project Proposal

We propose a project which uses a genetic algorithm to search for a better grammar. Genetic algorithms are a search technique inspired by evolution, taking a population of possible solutions and evaluating them with respect to some criteria [3]. Those which are successful are given more chances to ‘reproduce’ than those who are less successful. These search algorithms have been used to great effect in applications such as parameter estimation [2] and building phylogenetic trees [5]. The project would take the following form:

Creating an evaluation for grammars

This would involve finding test sets of various types of RNA with known secondary structures (for instance similar to the ones in [4]). There must be a training set from which parameters, such as the probability of production rule application, can be estimated using inside-outside training as detailed in [13]. The grammar being tested would then try to predict the secondary structure using the CYK algorithm, and this could be scored against the known secondary structure.

Creating an evolutionary/mutational model for grammars

The strength of the genetic algorithm is that grammars have the opportunity to change with each reproductive cycle. There are many possible evolutionary models for grammars and it is not immediately intuitive which would produce good results in this context. It is likely that during the course of the project this model is re-examined and modified to change the characteristics of grammars produced, so a broad model is chosen initially which has little restriction on its movements. In any case a starting model is required and so we used the following evolutionary/mutational system.

We begin by considering grammars in Chomsky Normal Form (CNF). Since for each grammar G there is an equivalent grammar G' in CNF [13], it is possible to produce all grammars in this form. After the benchmarking test is applied, grammars will reproduce/mutate:

Mutational model

A grammar G consisting of non-terminal variables $S = V_0, V_1, \dots, V_n$ and terminal variables a_1, \dots, a_m mutates with one of the following ways:

- The start variable (and corresponding production rules) change.
- A production rule is deleted at random
- A production rule of the form $V_i \rightarrow V_j V_k$ is added at random
- A production rule of the form $V_i \rightarrow a_j$ is added at random
- A new terminal or non-terminal variable is added
- A production rule of the form $V_i \rightarrow V_j V_k$ is changed to $V_i \rightarrow V_j V_l$, $V_i \rightarrow V_l V_k$ or $V_i \rightarrow V_l V_p$

Breeding model

Between generations, as well as mutating the population, and weeding out unsuccessful grammars, we may want to breed successful ones. This will allow large moves in the search space, while still maintaining desirable features. The question is then, what constitutes a good feature in a grammar, a single production rule is unlikely to add to the effectiveness of the grammar on its own.

Different grammars may be more effective at producing different part of the secondary structure. One way of producing the “offspring” of two strong grammars might then be to nest one inside the other, that is, having a non-terminal variable in one grammar lead to the production rules of the starting character, and adding all the production rules of the second, to the first.

Implementing the genetic algorithm

There are of course, many factors that must be considered before one can hope the the mutational and breeding models described above lead to a strong grammar, for example:

- *Probabilities of each evolutionary parameter.* With what probability might, say, a production rule be deleted at random? This probability could be kept with the grammar as it evolves, making it more or less likely to happen based on whether or not it was successful the last time it was implemented. The probability could reflect the whole population, for example use a weighting system based on the number of production rules of certain forms currently used.
- *Local stability of successful solutions.* It is possible that, given a strong grammar, a single mutation as described above will make it significantly less useful. The mutational model was designed to avoid this, but it could be that finding strong grammars is difficult in that evolving to them would take a step which was unlikely to happen, simply due to the size of the space of grammars.
- *Speed of evolution.* Given the size of the space of grammars and the relative size of the space of grammars that might be useful here, one might want to search slowly around the space to find these diamonds in the rough. Equally though, if one does not allow searching in larger areas, one might miss very strong grammars. This must be controlled during the search; for instance allowing multiple mutations in one generation would increase speed and modifying the weight of the fitness function might force ‘bad grammars’ out faster.
- *Null production rules.* The grammars produced could have production rules which have estimated probabilities close to or exactly zero. Does one continue to include these if the grammar moves forward? Eliminating them would allow a smaller number of parameters but might restrict searching movement.
- *The initial population.* Given the large space of possible grammars, it may be that the initial population limits the possible output of the algorithm. While the set of grammars in Fig.1 may make a good starting point, it is possible that a randomised initial population (for example multiply mutated versions of those in Fig.1) may lead to stronger output.
- *Starting from grammars not in CNF.* There are many other forms of grammar representation such as Embedded Normal Form and Lukas Normal Form [12] which could be used in a similar evolutionary model; one might expect different grammars to be searched through at different times if one is using a different representation.

One particular consideration is computational cost. Allowing grammars of arbitrary complexity will make the optimization of their probabilities, and implementation of CYK extremely costly steps in each generation. This could be limited by specifying a maximum number of production rules contained in a single grammar. One of the key benefits of use of grammars is their simplicity. Ideally we should look for similarly simple solutions. Another alternative would be to add a factor dependant on the number of production rules to the fitness function. This is more satisfactory, as it allows increasing complexity if it is shown to be sufficiently useful.

Other factors will impact on running time, including number of generations and population size. These have more obvious impact, so compromise between effectiveness and cost will be easier.

Representation of the grammar

There are many other forms of grammar representation which could be used in a similar evolutionary model; one might expect different grammars to be searched through at different times if one is using a different representation. Grammar forms which may be considered include:

- *Chomsky Normal Form* A grammar with production rules of the form
 - $A \rightarrow BC$ for $A \in N$ $B, C \in N \setminus \{S\}$
 - $A \rightarrow a$ for $A \in N$, $a \in V$
 - $S \rightarrow \epsilon$, where S is the start symbol and ϵ the empty string
- *Lucas Normal Form* [12] A grammar with production rules of the form
 - $A \rightarrow BC$ for $A \in N$ $B, C \in N \cup V$
- *Canonical Two Form* A grammar with production rules of the form
 - $A \rightarrow BC$ for $A \in N$ $B, C \in N \setminus \{S\}$
 - $A \rightarrow B$ for $B \in N \setminus \{S\}$
 - $A \rightarrow a$ for $A \in N$, $a \in V$
 - $S \rightarrow \epsilon$, where S is the start symbol and ϵ the empty string
- *Extended Greibach Normal Form* [7] A grammar with production rules of the form
 - $A \rightarrow aB_1B_2 \dots B_t$ for $A \in N$ $B_1, B_2, \dots, B_t \in N \setminus \{S\}$, $a \in V$
 - $A \rightarrow B$ for $B \in N \setminus \{S\}$
 - $A \rightarrow a$ for $A \in N$, $a \in V$
 - $S \rightarrow \epsilon$, where S is the start symbol and ϵ the empty string

where N the set of non-terminal symbols, V the set of terminal symbols that is disjoint from N and distinguished symbol $S \in N$ that is the start symbol. The evolutionary model can then be adapted for each normal form to search the space in a different manner.

Ambiguity

Dowell and Eddy (2004) note that while it is important to have ambiguous grammars, having *structurally ambiguous* grammars (grammars that give an identical secondary structure for two different derivations) can be undesirable. Evolving the grammars in the way proposed above may in fact lead to either unambiguous grammars or structurally ambiguous grammars. However, this is merely an artifact of the process; determining whether the grammar is ambiguous or not is undecidable [6]. One would hope that grammars which were problematic would perform poorly in the fitness tests and thus be filtered out by the evolutionary process, leaving the non-problematic ones.

Data

Two types of data will be used to test the algorithm. Firstly, artificial data will be created from a known grammar to test the method via simulation. This will enable shortcomings to be addressed as well as intuition gained for when real data is encountered. A grammar could be used, for example, that is not in CNF, and a method implemented which forces the grammar to be in CNF will be tested for accuracy of prediction. There are many other possibilities like this.

References

- [1] Andronescu, M., Bereg, V., Hoos, H., & Condon, A. (2008). RNA STRAND: The RNA Secondary Structure and Statistical Analysis Database. *BMC Bioinformatics*, 2008;9(1):340.
- [2] Cantone, I., Marucci, L., Iorio, F., Ricci, M., Belcastro, V., Bansal, M., Santini, S. et al. (2009). A yeast synthetic network for in vivo assessment of reverse-engineering and modeling approaches. *Cell*, 137, 172181.
- [3] Crosby, J. (1973). *Computer Simulation in Genetics*. London: John Wiley & Sons.
- [4] Dowell, R. & Eddy, S. (2004). Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. *BioMed Central Bioinformatics*, 5, 71-85.
- [5] Hill, T., Lundgren, A., Fredriksson, R., Schiöth, H. (2005). Genetic algorithm for large-scale maximum parsimony phylogenetic analysis of proteins. *Biochimica et Biophysica Acta*, 1725, 1929.v
- [6] Hopcroft, J. & Ullman, J. (1979). *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley.
- [7] Koch, R. & Blum, N. (1997). Greibach Normal Form Transformation, Revisited. Reischuk, Morvan (Eds.), STACS97 Proc., Lecture Notes in Computer Science, vol. 1200, Springer, New York, 1997, pp. 47-54.
- [8] Knudsen, B. & Hein, J. (1999). RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. *Bioinformatics*, 15, 456-454.
- [9] Knudsen, B. & Hein, J. (2003). Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Research*, 31, 3423-3428.
- [10] Krogh, A., Brown, M., Mian, I., Sjölander, K. & Haussler, D. (1993). Hidden Markov Models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235, 1501-1531.
- [11] Lefebvre, F. (1996). A grammar-based unification of several alignment and folding algorithms. *ISMB-96 Proceedings*, 2, 143-154.
- [12] Lukas, L. (1994). Structuring Chromosomes for Context-Free Grammar Evolution. *15th International Conference On Evolutionary Computing*, 130- 135.
- [13] Lyngsø, R. (2009). Department of Statistics website, retrieved from http://www.stats.ox.ac.uk/___data/assets/pdf_file/0008/4778/grammatical_models.pdf on 12/12/2009.
- [14] Rabiner, L. (1989). A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE* 77, 2, 257286.
- [15] Rivas, E. & Eddy, S. (2000). The language of RNA: a formal grammar that includes pseudoknots. *Bioinformatics*, 16, 334-340.
- [16] Sakakibara, Y, Brown, M., Hughey, R., Mian, I., Sjolander, K., Underwood, R. et al. (1994). Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Research*, 22, 5112-5120.