GENETIC ALGORITHMS FOR STOCHASTIC CONTEXT-FREE GRAMMAR PARAMETER ESTIMATION

by

KAAN TARIMAN

(Under the Direction of Liming Cai)

ABSTRACT

Stochastic grammar models for biological sequences have been extensively used in secondary structure prediction and profiling for structural homology recognition. A pertinent issue is, given training RNA sequences, how to estimate the stochastic parameters associated with the rules in the grammar efficiently and accurately. In particular, the existing algorithms for parameter estimation, such as Inside-Outside, have local maxima and time complexity problems.

We introduce a genetic algorithm method to solve the parameter estimation problem. Being global optimization methods, genetic algorithms do not suffer from the locality problem and they are scalable and flexible. The model uses an evaluation function that calculates the maximum likelihood to generate a sequence given a parameter set of a grammar. Our experiments with the implemented algorithm demonstrate its effectiveness in parameter estimation for specific grammar models based on both simple RNA structures and tRNA sequences.

INDEX WORDS: genetic algorithms, stochastic context-free grammars, RNA secondary structure prediction, parameter estimation.

GENETIC ALGORITHMS FOR STOCHASTIC CONTEXT-FREE GRAMMAR PARAMETER ESTIMATION

by

KAAN TARIMAN

B.S in Computer Engineering, Bogazici University, Turkey, 2002

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2004

GENETIC ALGORITHMS FOR STOCHASTIC CONTEXT-FREE GRAMMAR PARAMETER ESTIMATION

by

KAAN TARIMAN

Major Professor: Liming Cai

Committee: Khaled Rasheed
Russell L. Malmberg

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May, 2004

This thesis is dedicated to my family

and Asli for their great emotional support.

ACKNOWLEDGEMENTS

I would like to thank Assoc. Prof. Liming Cai, Assoc. Prof. Khaled Rasheed and Prof. Russell Malmberg for serving on my advisory committee and their support. I would especially like to thank my advisor, Prof. Cai, for his guidance, encouragement, support and inspiration. Many thanks to our research group; I have learned a lot from my colleagues when I first joined the group. I would like to thank all the group members for their help, especially to Jizhen Zhao, Yinglei Song and Chunmei Liu, for helping me out with the systems they have developed for our group and letting me use them.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS V

LIST OF FIGURES VII

1. INTRODUCTION 1

2. RNA FOLDING PROBLEM 5

2.1. DEFINITIONS 5
2.2. RNA SECONDARY STRUCTURES WITH PSEUDOKNOTS 9
2.3. SCFG APPROACH 10
2.4. INSIDE-OUTSIDE ALGORITHM 11
2.5. CYK ALGORITHM 14

3. EC METHODS IN BIOINFORMATICS 16

3.1. INTRODUCTION 16
3.2. PROTEIN FOLDING PROBLEM (PFP) 17
3.3. USING PARALLEL FAST MESSY GENETIC ALGORITHMS 24
3.4. RNA FOLDING PROBLEM 26
3.5. EVOLVED NEURAL NETWORKS 30
3.6. CONCLUDING REMARKS 33

4. THE GENETIC ALGORITHM MODEL 36

4.1. GENETIC ALGORITHM COMPONENTS 36
4.2. ENCODING 38
4.3. FITNESS FUNCTION 40
4.4. SELECTION AND REPLACEMENT METHODS 41
4.5. GENETIC OPERATORS 42

5. EMPIRICAL AND THEORETICAL ANALYSIS 44

5.1. SIMPLE SEQUENCE ANALYSIS 44
5.2. COMPLEX SEQUENCE ANALYSIS 54
5.3. TERMINATION CONDITION 56
5.4. GRAMMAR CONSTRUCTION 58
5.5. TIME ISSUES 60

6. CONCLUSION 63

6.1. CONCLUDING REMARKS 63
6.2. FUTURE DIRECTIONS 64

7. REFERENCES 67

APPENDIX A 71

LIST OF FIGURES

FIGURE 1 : *BACILLUS SUBTILIS* RNASE P RNA AND ITS USUAL DISPLAY OF SECONDARY STRUCTURES. 6
FIGURE 2 : *S1* AND *S2* HAVE THE SAME STRUCTURE BUT NOT *S3*. 8
FIGURE 3 A PARSE TREE FOR *S1* USING THE EXAMPLE GRAMMAR 9
FIGURE 4 : CALCULATION OF $\alpha(i, j, v)$ IN INSIDE ALGORITHM (DURBIN ET AL., 1998) 12
FIGURE 5 : CALCULATION OF ITERATION STEP IN OUTSIDE ALGORITHM (DURBIN ET AL., 1998) 13
FIGURE 6 : A CONFORMATION OF A CHAIN ON THE TWO-DIMENSIONAL LATTICE. (A-B) AN EXAMPLE OF A $+90^{o}$ ROTATION MUTATION. NOTE -90^{o} IS NOT SELF-AVOIDING. OTHER SIMPLE MUTATIONS ARE SHOWN IN (C-E). (FOGEL AND CORNE 2003) 19
FIGURE 7 : LOWEST ENERGY STATE OF A 64-RESIDUE CHAIN. BLACK BLOCKS ARE HYDROPHOBIC RESIDUES. KONIG AND DANDEKAR (1999) 20
FIGURE 8: (Θ, Φ) PAIR IN A SEQUENCE OF FOUR RESIDUES 21
FIGURE 9: MEAN NUMBER OF GENERATIONS REQUIRED UNTIL CONVERGENCE. MEAN WAS CALCULATED OVER 20 RUNS. ERROR BARS FOR DATA POINTS ARE SHOWN TOO. (SHAPIRO ET AL., 2001) 27
FIGURE 10: IMPROVEMENTS IN THERMODYNAMIC STABILITY AS A FUNCTION OF POPULATION SIZE FOR FODINGS OF POLIO 3 AND PSTVD. (SHAPIRO ET AL., 2001) 28
FIGURE 11 : TABULAR REPRESENTATION FOR A SEQUENCE OF LENGTH *N*. 29
FIGURE 12 : THE GENERATION PROCESS IN A TYPICAL GA ITERATION 38
FIGURE 13 : OUTPUT FOR SIMPLE SEQUENCE *W1* WITH A GENERAL GRAMMAR. 46
FIGURE 14: COMPARISON OF THE ORIGINAL AND PREDICTED OUTPUT WHEN A GENERIC GRAMMAR MODEL IS USED. 47
FIGURE 15 : OUTPUT FOR SIMPLE SEQUENCE *W1* WITH A SPECIFIC GRAMMAR. 48
FIGURE 16 : OUTPUT FOR SIMPLE SEQUENCE *W2* WITH S SPECIFIC GRAMMAR. 50
FIGURE 17 : COMPARISON CHART FOR GA AND EM 51
FIGURE 18 : A PART FROM THE OUTPUT OF CYK TRACEBACK 51
FIGURE 19 : OUTPUT FOR A TWO-STEM SEQUENCE WITH A GENERAL GRAMMAR. 53
FIGURE 20 : A) PREDICTED STRUCTURE B) ORIGINAL STRUCTURE 53
FIGURE 21 : TRNA GENERAL STRUCTURE 55
FIGURE 22 : ANNOTATION OUTPUT FOR ALA TRNA 56
FIGURE 23 : A) PREDICTED DERIVATION TREE OF ALA TRNA B) ORIGINAL DERIVATION TREE FOR THE SAME MOLECULE 56
FIGURE 24: FITNESS VALUES CONVERGING THROUGH GENERATIONS 57

FIGURE 25 : NUMBER OF GENERATIONS UNTIL GA CONVERGES 58
FIGURE 26 : ACCURACY OF PREDICTION AS A FUNCTION OF POPULATION SIZE. 61

1. INTRODUCTION

RNA (ribonucleic acid) secondary structure is similar to an alignment of nucleic acid sequences, except that the sequence folds back on itself and forms complementary base pairs. Many interesting RNAs conserve a secondary structure of base-pairing interactions more than they conserve their sequence. This makes RNA sequence analysis complicated and difficult.

Simple RNA molecules are composed of four basic nucleotides (also known as bases), namely Adenine (A), Cytosine (C), Guanine (G), and Uracil (U). Despite their biochemical resemblance to DNAs, RNAs have very different biological functions due to their structures. Unlike the double-stranded DNAs that form double helical structures, RNAs are usually single-stranded molecules, and hence they fold by forming pairs of bases to produce stable structures that minimize their energy level.

The complementary bases, C-G and A-U form stable base pairs with each other through the creation of hydrogen bonds between donor and acceptor sites on the bases. These are called ``Watson-Crick (W-C)" base pairs. In addition, we consider the weaker G-U wobble pair, where the bases bond in a skewed fashion. All of these are called canonical base pairs. Other base pairs may occur, some of which are stable. They are called non-canonical base pairs.

RNA secondary structure prediction problem is considered mainly from two different approaches, thermodynamics and formal grammars. In the former, the problem of predicting the optimal secondary structures of RNA sequences is to find, given some energy parameters, the optimal way to pair up the bases of the input RNA so as to minimize the overall energy level of the RNA. One of the first algorithms that solves a simple version of this problem is Zuker's Algorithm (Zuker, 1989). Later, Rivas and Eddy proposed a new algorithm for a much wider class of problems in this domain including pseudoknots (Rivas and Eddy, 1999).

The other approach to the prediction problem is grammatical models. Since RNA sequences are not just linear sequences of ribonucleic acids randomly generated but sequences intended to convey genetic information of specific biological functions (or meanings), it appears quite reasonable to assume the existence of a certain kind of grammatical devices by which RNA sequences can be generated and modeled.

Therefore, in order to solve the prediction problems concerning biological structures such as RNA secondary structures, protein secondary (or tertiary) structures, no matter what structures they are, it may be of great importance to propose certain formal systems (grammars, automata, etc.) to model those structures in an appropriate manner. This enables us to analyze the biological properties of RNA, DNA or amino acid sequences in terms of formal language theoretic concepts (Searl, 1993).

RNA secondary structure prediction has become an important problem in bio-informatics while using stochastic context free grammars (SCFG) for modeling the structures has become popular in the last decade since it was first proposed by Sakakibara et al. (1994). Under this topic, one challenging problem is learning the grammar rules and probability distributions associated with the set of rules. The Inside-Outside (IO) Algorithm is a well known estimator for learning SCFG parameters from a training set (Lari & Young, 1990). This algorithm is an implementation of the expectation maximization (EM) method for SCFG where satisfactory parameters are obtained through an iterative re-estimation process.

The IO algorithm has two major problems. First, local maxima are much more of a problem than in its counterpart, the forward-backward algorithm for Hidden Markov Models (HMMs). Second, it is slow and has $O(L^3M^3)$ time complexity, where L is the length of the sequence and M is the number of non-terminals. In this paper we focus on how to solve these two problems.

Here, we introduce a new approach for parameter estimation of such grammars based on the idea of genetic algorithms (GA) (Holland, 1975), a non-deterministic optimization procedure. Genetic Algorithms are derived from the concept of biological evolution using the operations based on the survival of the fittest, recombination (crossover), mutation and selection to mimic genetic processes. They have been applied in a wide variety of fields in science and engineering (Goldberg, 1989; Holland, 1992) and they have been developed for RNA sequence folding (Shapiro and Navetta, 1994; Shapiro and Wu, 1996, 1997; Shapiro *et al.*, 2001). These genetic algorithms and other evolutionary approaches have been focused on sequence analysis and deriving stem-loop structures with minimal energies. In our approach we focus on estimating parameters for grammatical modeling of those structures.

The idea for this approach comes from the fact that given an SCFG model, we can find the likelihood of a sequence. Therefore GA can find the optimum parameter set in the search

space by comparing them with the likelihood values. The fitness value is calculated by the Inside Algorithm (Durbin *et al.*, 1998) which gives the total probability to generate a given sequence with the given SCFG.

GAs have many advantages over other search techniques in complex domains such as the RNA folding problem. They tend to avoid being trapped in local optima and can handle both discrete and continuous optimization variables such as in our case where the set of rules have continuous parameters. The performance of a GA can be measured by how well it explores the search space and at the same time how well it exploits the potential regions for optima. A good GA explores the search space widely in the early generations and then starts exploiting the optima until it converges to various optima as generations go. Our experiments with this implementation demonstrate the accuracy and effectiveness of the GA approach in parameter estimation for SCFG models in RNA folding problem.

Our new approach is very likely to find the global optimum through non-deterministic optimization. The algorithm is simple, involving nothing more complex than copying strings, swapping partial strings and doing basic arithmetic. The time complexity of the algorithm can be found by multiplying the complexity of the fitness function by number of non-terminals since the length of the genome is proportional to that number. This results in $O(L^3M^2)$ time complexity where M is the number of nonterminals and L is the length of the sequence. This is better than inside-outside algorithm which has $O(L^3M^3)$ complexity and the constants are scalable in our case. The advantage of our framework also comes from its generality since the fitness function can be changed to any grammatical probabilistic modeling system including pseudoknot structures (Cai *et al.*, 2003) or any other likelihood determination method.

As described in the literature (Goldberg, 1989), the original GAs used a fixed binary-string representation (genotype) for the population of objects (referred to as individuals) that evolved towards fitness. However, our GA for parameter estimation has a floating point number string representation, storing the probabilities associated with the set of rules. The GA attempts to find a good parameter set by randomly generating a collection of potential solutions and then manipulating those solutions using genetic operators. Those operators use existing solutions to produce new solutions. Each solution is assigned a fitness value which is the logarithm of the total probability or likelihood to generate the given RNA sequence. The key idea is to select for reproduction the solutions with higher fitness and apply the genetic operators to them to generate

new individuals. The newly generated individuals may give better fitness values through mutation and re-combination operations and evolve towards various optima possibly including the global optimum. The process usually continues until an acceptable solution is found, usually we bound the number of generations with certain value.

The individuals are initialized randomly. Each individual has the same number of genes as the number of the rules in the grammar to be optimized. Since these values are probabilities associated with the rules, a gene can only have a value from the range [0, 1]. Another constraint for the values is that the sum of all probabilities associated with the same non-terminal on the left hand side must be 1. Therefore our system repairs the individual after the mutation and re-combination operations if the constraint does not hold.

Each generation is replaced with a specific percentage where most fir individuals are kept for the next iteration. This replacement method is used in Steady-State GAs (our approach) which converge faster than Generational GAs where the entire population is replaced by a new generation. The recombination method is chosen as Heuristic Crossover, a recombination process producing an offspring closer to the better parent. This is a greedy operator moving in the search space in the direction of the parent with the greater fitness value. Our model uses Gaussian Mutation which is a suitable method for any floating point number genotype as a mutation operation. Selection for recombination process is done by Roulette Wheel Selection where an individual has the chance to carry its genes through generations proportional to its fitness value.

The layout of this thesis starts from introducing the problem domain where we explain the RNA folding problem and the approaches used so far (Chapter 2). Then we conduct a survey of evolutionary computation and its applications in similar problems in bioinformatics (Chapter 3). We finally introduce our GA system and its components (Chapter 4). The results of the system, empirical and theoretical analysis with couple of example grammar and sequence examples are shown in the next chapter (Chapter 5). Finally we make our concluding remarks and future research suggestions for extending and modifying our work (Chapter 6).

2. RNA FOLDING PROBLEM

2.1. Definitions

RNA is a single-stranded nucleic acid made up of 4 types of nucleotides; adenine, uracil, cytosine and guanine. RNA is involved in the transcription of genetic information and also several biochemical reactions.

2.1.1. RNA Terminology

The secondary structure of an RNA molecule is the collection of base pairs that occur in its 3 dimensional structure. An RNA sequence can be represented as

$$X = x_1, x_2, x_3, ..., x_n$$

where x_i is called the ith (ribo)nucleotide. Each x_i belongs to the set {A,C,G,U}. We will refer to i as the ith base in the sequence. A secondary structure, or folding, on X is a set S of ordered base pairs, written as $i.j$, $1 \le i < j \le n$ satisfying:

1. $j - i > 3$

2. If $i.j$ and $i'.j'$ are 2 base pairs, (assuming without loss in generality that $i \le i'$), then either:

(a) $i = i'$ and $j = j'$ (they are the same base pair),

(b) $i < j < i' < j'$ (parallel stem-loop), or

(c) $i < i' < j' < j$ (nested stem-loop).

(d) $i < i' < j < j'$ (pseudoknots)

In Figure 1 : *Bacillus subtilis* RNase P RNA and its usual display of secondary structures., an RNA with its secondary structures is indicated with labels. The letters within the structure stand for:

M : multi-loops

I : interior loops

B: bulge loops

H: hairpin loops (stem-loops)

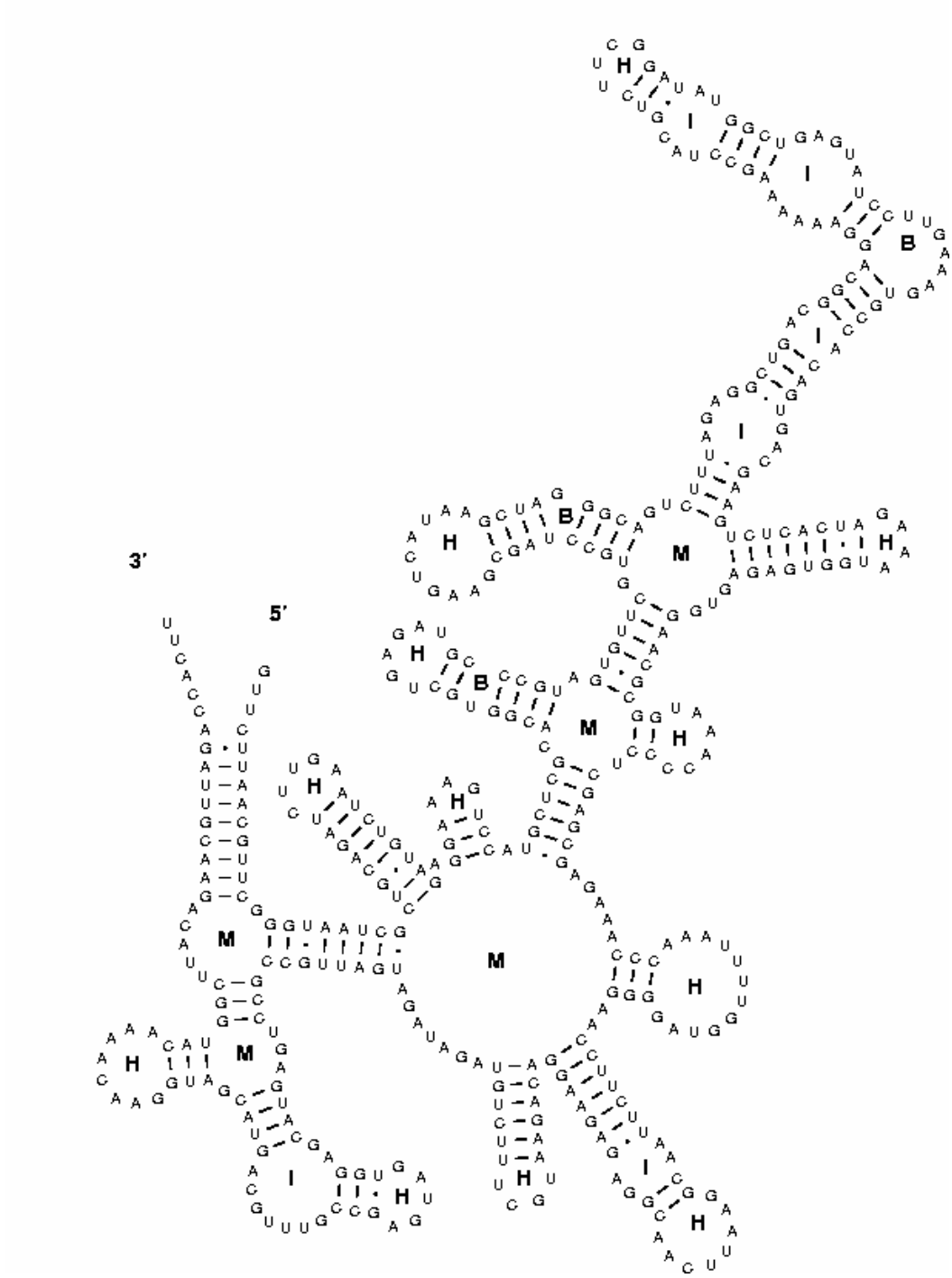


Figure 1 : *Bacillus subtilis* RNase P RNA and its usual display of secondary structures.

2.1.2. Context Free Grammars

A general theory for modeling strings of symbols has been developed by computational linguists. This theory is known as the Chomsky hierarchy of transformational grammars (Chomsky, 1959). The theory was developed in an attempt to understand the structure of the natural languages. They became important in theoretical computer science (Hopcroft & Ullman, 1979) because computer languages, unlike natural languages, can be precisely specified as formal grammars.

A formal grammar consists of a finite set of *symbols* and a set of rewriting rules $\alpha \to \beta$ (also called *productions*) where α and β are both strings of symbols. There are two kinds of symbols: abstract *nonterminal* symbols and *terminal* symbols that can appear in an observed string. The left-hand side α contains at least one nonterminal, which in general is transformed into a new string of terminals and/or nonterminals on the right-hand side of the production. In modeling molecular structures terminals are a set of amino-acid or nucleotide symbols.

In the following examples, we use W to represent any nonterminal, a to represent any terminal, α and γ to represent any string of nonterminals and/or terminals including the null string, and β to represent any string of nonterminals and/or terminals not including the null string.

Regular Grammars: Only production rules of the form $W \to aW$ or $W \to a$ are allowed.

Context-free Grammars : Any production rule of the form $W \to \beta$ is allowed. The left-hand side of the production rule must consist of just one nonterminal but the right-hand side can be any string.

Context-sensitive Grammars: Productions are of the form $\alpha_1 W \alpha_2 \to \alpha_1 \beta \alpha_2$. The allowed transformations of nonterminal W are dependent on its context α_1 and α_2. It is provably equivalent to require that the right-hand side contains at least as many symbols as the left-hand side; context-sensitive grammar productions never shrink.

Unrestricted Grammars : Any production of the form $\alpha_1 W \alpha_2 \to \gamma$ is allowed.

Since we will be dealing with the context-free grammars, here we present the formal definition of them:

A context-free grammar (CFG) is defined by a quadruple $G = (N, \Sigma, P, S)$, where *N,* is an alphabet of *nonterminal symbols*, Σ is an alphabet of *terminal symbols* such that $N \cap \Sigma = \emptyset$, *P* is a finite set of production rules of the form $W \to \alpha$ for $W \in N$ and $\alpha \in (N \cup \Sigma)^*$, and *S* is a special nonterminal called the *start symbol*. The *language generated* by a CFG *G* is denoted *L(G).*

A *stochastic context-free grammar* (SCFG) *G* consists of a set of nonterminal symbols *N,* a terminal alphabet Σ, a set *P* of production rules with associated probabilities, and the start symbol *S*. The associated probability for every production $W \to \alpha$ in *P* is denoted Pr($W \to \alpha$), and for every nonterminal, a probability distribution exists over the set of productions have the same nonterminal on the left-hand side.

In bio-informatics, HMM approaches and other sequence alignment methods can also be classified as the lowest type under the hierarchy explained above. On the other hand context-free grammars permit additional rules that allow the grammar to create nested, long-distance pair wise correlations between terminal symbols. Since RNA secondary structure conservation does not imply sequence conservation, context-free grammars are suitable to model them. We can have a better idea about the representation with the example below.

In Figure 2 (Durbin et al., 1998) , *s1* and *s2* can share same RNA secondary structure although they have different sequences because they share the same pattern of base pairs (A-U and C-G). *s3*, inherits its sequence from the first half of *s2* and the second half of *s1*, cannot fold into a similar structure.

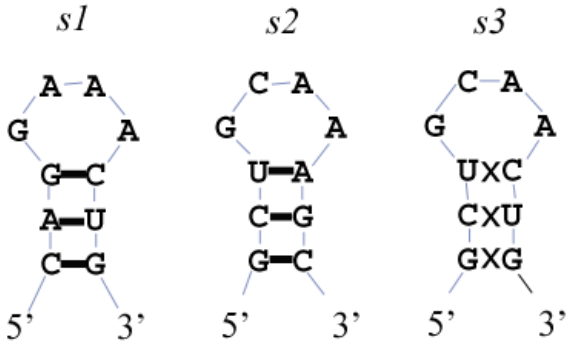


Figure 2 : *s1* and *s2* have the same structure but not *s3.*

Below there is a CFG given for the specific structure shown in Figure 2. Notice the rule structures where the base pairing can be represented in a trivial way (Durbin et al., 1998).

$$S \to aW_1u \mid cW_1g \mid gW_1c \mid uW_1a,$$
$$W_1 \to aW_2u \mid cW_2g \mid gW_2c \mid uW_2a,,$$
$$W_2 \to aW_3u \mid cW_3g \mid gW_3c \mid uW_3a,$$
$$W_3 \to gaaa \mid gcaa.$$

When a sequence is derived by the context-free grammar, the parse (or derivation) tree can be represented by a planar graph since the pairing regions are either parallel or nested. Using the example grammar and the sequence *s1* we can draw the parse tree as follows (Figure 3 , Durbin et al., 1998).

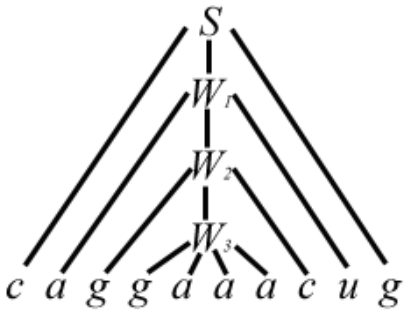


Figure 3 A parse tree for *s1* using the example grammar

2.2. RNA Secondary Structures with Pseudoknots

We have seen the secondary structure and the related terminologies in the previous section. In this section we want to emphasize the importance of these structures.

RNA was once thought to be the passive intermediary messenger between DNA genes and the protein translation machinery. These types of RNA are called **coding RNA's**. However, many non-coding RNA's exist which adopt sophisticated three-dimensional structures, and some even catalyze bio-chemical reactions. Proteins show similar properties in the sequence of the

amino-acids namely the polypeptide chains. Modeling proteins is discussed in more detail in Chapter 3.

When the parts of the RNA sequence spanned by two base pairs are neither disjoint, nor have one contained in the other, the two base pairs form a **pseudoknot**. One reason that the RNA secondary structure prediction community has been able to get away with ignoring pseudoknots for nearly thirty years is that for known true RNA structures you can usually find a set of at least 95% of the base pairs that does not contain any pseudoknots; on the other hand, almost all RNA structures contain one or more pseudoknots. Therefore one way to try to improve on prediction accuracy would be to find a feasible way to include structures with pseudoknots. Our proposed system can deal with this case without any difficulty unless there is a likelihood method dealing with pseudoknots (i.e Parallel Communicating Grammar Systems proposed by Cai et al., 2003).

2.3. SCFG Approach

Once we decide to model the sequences by using stochastic context-free grammars, we must also have algorithms to address the three basic problems:

i) The alignment problem: finding an optimal alignment of a sequence to a parameterized stochastic grammar.

ii) The scoring problem: finding the probability of a sequence given a parameterized stochastic grammar.

iii) The training problem: given a set of sequences, estimate optimal probability parameters for an unparameterised stochastic grammar. In this paper we are focusing on this problem.

The same problems are applicable for modeling with HMMs. Briefly, Viterbi algorithm solves the alignment problem, forward pass of the forward-backward algorithms solves the scoring problem and forward-backward algorithm is used in Baum-Welch expectation maximization to address the training problem (Baum, 1972). Actually these dynamic programming algorithms are analogous for stochastic context-free grammars. The analogy can be formed for Inside and CYK algorithms with forward and Viterbi algorithms respectively.

From Section 2.1.2 we have seen that context free grammars can have an unlimited variety of symbol strings on the right-hand side of their production rules. To express a general CFG parsing algorithm, it is very useful to adopt a restricted 'normal form'. One such normal form is *Chomsky normal form*. This normal form requires that all CFG production rules are of the form $W \to YZ$ or $W \to a$. It is very important to state that any CFG can be converted into this normal form without changing the language it generates. The disadvantage of conversion is having more non-terminals which results with more computation in any dynamic programming approach. The advantage on the other hand is handling the algorithm in a simpler way.

2.4. Inside-Outside Algorithm

This algorithm is basically a counter-part of the forward-backward algorithm for HMMs calculating the probability (score) of a sequence given an SCFG. It is a recursive dynamic programming algorithm however the computational complexity is substantially greater than forward-backward.

The inside algorithm calculates the probabilities $\alpha(i, j, v)$ of a parse sub-tree rooted at non-terminal W_v for sub-sequence $x_i,...x_j$ for all *i, j* and *v* (Figure 4). The outside algorithm calculates a probability $\beta(i, j, v)$ of a complete parse tree rooted at the start non-terminal for the complete parse rooted at the start non-terminal for the sub-sequence $x_i,...x_j$ rooted at non-terminal W_v for all *i, j* and *v*. The Expectation Maximization (EM) algorithm is based on iterative optimization that will converge to parameter values at a local maximum of the likelihood function. Here likelihood is the probability of data given the model, i.e., *P(D|M)* which is often used to indicate how good the model predicts the data.

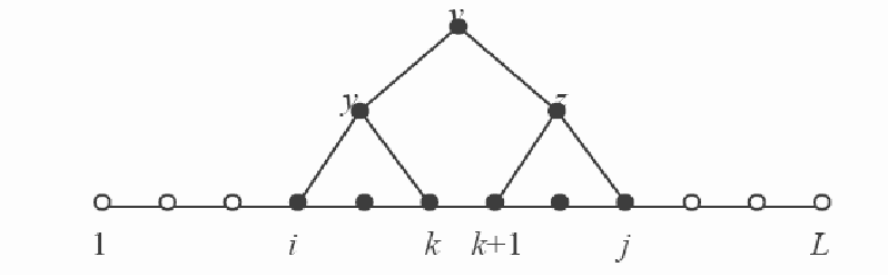


Figure 4 : Calculation of $\alpha(i, j, v)$ in Inside Algorithm (Durbin et al., 1998)

Inside Algorithm (Durbin et al., 1998) :

Initialization: (for i=1 to L, v=1 to M)

$$\alpha(i,i,v) = e_v(x_i).$$

Iteration: (for i=1 to L, v=1 to M)

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

Termination: $P(x \mid \theta) = \alpha(1, L, 1)$.

Here the iteration step calculates the probability of the parse sub-tree rooted at non-terminal W_v for the sub-sequence from i to j is denoted by $\alpha(i, j, v)$. This value is calculated recursively by summing the probabilities of parse sub-tree rooted at non-terminal W_y and W_z for smaller sub-sequences i to k and $k+1$ to j, for all y, z and k, weighted by the probability $t_v(y, z)$ which is the probability associated with the rule $V \to YZ$.

Outside Algorithm (Durbin et al., 1998):

Initialization: (for i=1 to L, v=1 to M)

$$\beta(1, L, 1) = 1;$$

$$\beta(1, L, v) = 0 \text{ for } v=2 \text{ to } M.$$

Iteration: (for i=1 to L, j=L to i, v=1 to M)

$$\beta(i,j,v) = \sum_{y,z} \sum_{k=1}^{i-1} \alpha(k,i-1,z)\beta(k,j,y)t_y(z,v) + \sum_{y,z} \sum_{k=j+1}^{L} \alpha(j+1,k,z)\beta(i,k,y)t_y(v,z).$$

Termination: $P(x \mid \theta) = \sum_{v=1}^{M} \beta(i,i,v)e_v(x_i)$ for any i.

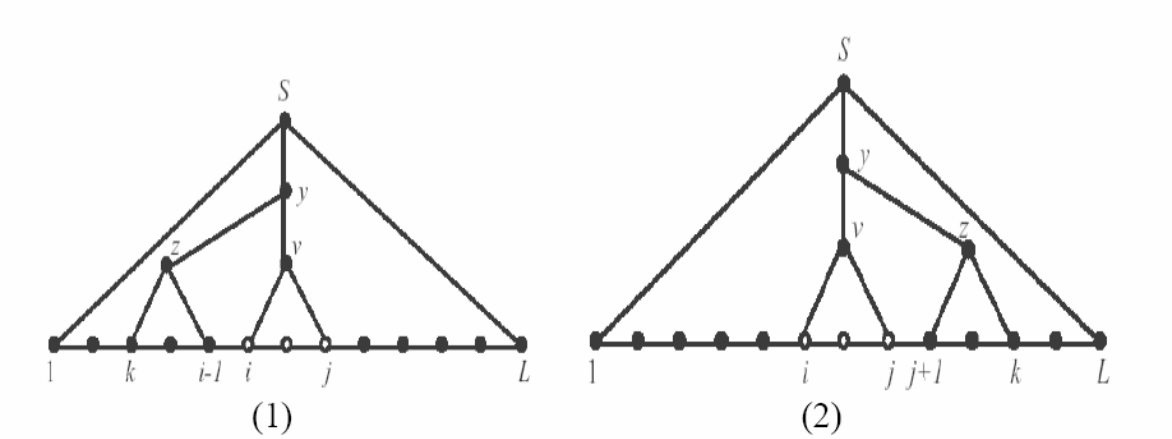


Figure 5 : Calculation of iteration step in Outside Algorithm (Durbin et al., 1998)

In Figure 5 the recursive calculation of $\beta(i, j, v)$, the probabilities of all parse trees excluding sub-trees rooted at non-terminal W_v for sub-sequence $x_i,...x_j$ are illustrated. Figure 5 (1) corresponds to the first part of the outside iteration which combines the outside value for non-terminal W_y and subsequence $1,...,k, j+1,...,L$, the inside value for non-terminal W_z filling in the sub-sequence $k,...,i-1$, and the probability $t_y(z,v)$. Figure 5 (2) corresponds to the second part of the outside iteration which combines the outside probability for non-terminal W_y on the excluded sub-sequence $i,...,k$, the inside probability for non-terminal W_z filling in the sub-sequence $j+1,...,k$, and the probability $t_y(v,z)$.

Inside-Outside Algorithm (Durbin et al., 1998):

The two algorithms explained above are combined to form the parameter re-estimation algorithm which is a type of EM method. The α and β variables are combined in the following equation to find the expected number of times that nonterminal v is used in a derivation:

$$c(v) = \frac{1}{P(x \mid \theta)} \sum_{i=1}^{L} \sum_{j=i}^{L} \alpha(i, j, v) \beta(i, j, v).$$

If we expand the equation above to find the expected number of times that W_v is reached in the derivation tree and the rule $W_v \rightarrow W_y W_z$ is used, we get:

$$c(v \to yz) = \frac{1}{P(x \mid \theta)} \sum_{i=1}^{L-1} \sum_{j=i+1}^{L} \sum_{k=i}^{j-1} \alpha(i,k,y)\alpha(k+1,j,z)\beta(i,j,v)t_v(y,z) .$$

The re-estimation is calculated by the ratio of these two expected values. Therefore the probability of the production rule $W_v \to W_y W_z$ in the SCFG is:

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

The probability parameter of other type of production rules in CNF such as $W_v \to a$ is calculated by:

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

The re-estimation equations can be simply extended in order to use multiple independent sequences other than only using a single observed sequence. The only extension is to sum expected values over all sequences in the training set.

2.5. CYK Algorithm

The optimal derivation tree of a sequence can be found once we have the SCFG model. This dynamic programming method namely Cocke-Younger-Kasami (CYK) algorithm is a variant of the Inside Algorithm with max operations replacing the sums. In the below definition $\gamma(i,j,v)$ represents the logarithm of the probability to generate the optimum parse tree rooted at v generating the sequence $x_i,\ldots,x_j$. This value is denoted by $\log P(x,\hat{\pi} \mid \theta)$ where $\hat{\pi}$ is the most probable parse tree. After we find the optimum tree we are using a traceback variable in order to keep which rules we have gone through while generating the sequence. This variable is denoted

by $\tau(i, j, v)$ which includes triplet of numbers *(y,z,k)* representing the three-dimensional location in the dynamic programming matrix. The algorithm is defined formally:

CYK Algorithm (Durbin et al., 1998):

Initialization: (for i=1 to L, v=1 to M)

$$\gamma(i,i,v) = \log e_v(x_i);$$
$$\tau(i, j, v) = (0,0,0).$$

Iteration: (for i=1 to L, v=1 to M)

$$\gamma(i, j, v) = \max_{y,z} \max_{k=i\ldots j-1} \{\gamma(i,k,y) + \gamma(k+1, j, z) + \log t_v(y,z)\};$$
$$\tau(i, j, v) = \arg\max_{(y,z,k),k=i\ldots j-1} \{\gamma(i,k,y) + \gamma(k+1, j, z) + \log t_v(y,z)\}.$$

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

Once the CYK algorithm finds the optimum derivation value for the given sequence, a traceback algorithm is called in order to recover the best alignment and determine the derivation tree. This algorithm is defined using the stack data structures:

CYK Traceback Algorithm (Durbin et al., 1998):

Initialization:

Push (1,L,1) on the stack.

Iteration :

Pop(i,j,v).

(y,z,k) = $\tau(i, j, v)$.

If $\tau(i, j, v)$ =(0,0,0) (implying i=j), attach x_i as the child of v.

Else : attach y,z to parse tree as children of v.

Push(k+1, j,z).

Push(i,k,y).

3. EC METHODS IN BIOINFORMATICS

3.1. Introduction

Biological sequence analysis is a highly sophisticated field which is sometimes regarded as a sub-field of bio-informatics. The field is highly related to both biology and computer science. The researchers are trying to employ different aspects of these two sciences in order to understand the biological sequences, the structures formed within them, the special regions and the bio-chemical functions related to them. Sequence analysis can be divided into two groups with respect to the type of sequence: The first is DNA analysis where the problem is to find a specific sub-sequence associated with a specific purpose. The second is RNA and protein analysis where the main purpose is to find the 3-dimensional structures they form when they are folding. In this chapter we will be focusing mostly on RNA and protein folding problem approached by evolutionary computation methods.

The fundamental principles underlying evolutionary computation can best be summarized by nineteenth century naturalist Charles Darwin, who was fascinated by the origin of the many complex forms of life existing in nature. In his introduction to The Origin of Species, Darwin (1859) made the following observation:

"As many more individuals of each species are born than can possibly survive; and as, consequently, there is a frequently recurring struggle for existence, it follows that any being, if it varies however slightly in any manner profitable to itself, under the complex and sometimes varying conditions of life, will have a better chance of surviving, and thus be naturally selected. From the strong principle of inheritance, any selected variety will tend to propagate its new and modified form."

To adapt these principles to problem solving, one can construct an evolutionary simulation in which the individuals represent alternative solutions to the target problem. The frequently recurring struggle for existence that Darwin observed in nature is inherent in our model due to the limited computer resources we can devote to our simulation. This is expressed

both in restrictions on the number of alternative problem solutions we store in computer memory, and the computational resources available for evaluating these solutions. Variation between individuals is a result of making random changes to the population of evolving solutions, and from recombining pieces of old solutions to produce new solutions. Darwin's process of natural selection can be modeled by imposing a selection distribution on the population of solutions such that the better ones have a higher probability of being recombined into new solutions and thereby preserving the attributes that made them viable. Alternatively, we can use this selection distribution to ensure that the poorer solutions have a higher probability of being replaced by new solutions. To determine how good a particular solution is, the evolutionary algorithm applies the solution to the target problem within the context of a domain model and evaluates its fitness through the use of appropriate metrics. This forms the main principle underlying evolutionary computation (EC).

In this chapter we conduct a brief survey primarily restricted to work performed on how evolutionary algorithms (EA) have been employed in the field of bioinformatics in the past decade (Clark and Westhead, 1996 has a similar one for the other decade).

3.2. Protein Folding Problem (PFP)

Organisms contain thousands of different types of proteins that are responsible for transporting small molecules (e.g hemoglobin transports oxygen in the bloodstream), catalyzing biological functions, providing structure to collagen and skin, regulating hormones, and many other functions. Each protein is a sequence of amino acids of 20 different types, bound into linear chains that adopt a specific folded three-dimensional shape. Each shape provides valuable clues to the protein’s function. Indeed, this information is essential to the design of the new drugs capable of combating disease. Non-coding RNA sequences (unlike messenger RNA’s that are used to encode amino acids, not including secondary structures) also display this kind of shape and structures as in proteins folding onto themselves.

Regrettably, predicting the shape of a protein is a difficult, expensive task, which explains why relatively few proteins have been categorized in this regard. Virtual protein

models, created on computer systems, may provide a cost-effective solution to the problem, trying to predict the structure of a protein only given the sequence of amino acids. This is a combinatorial optimization problem, which has exponential number of potential solutions. The analogy can again be formed for RNA's where the problem is the structure prediction given the sequence of nitrogenous based nucleotides: adenine (A), guanine (G), thymine (T) and cytosine (C).

The *primary structure* of a protein's polypeptide chain is its sequence of amino acids. Different regions of this sequence tend to form regular, characteristic shapes called *secondary structures*. The three main categories of secondary structures are the α-helix β-strand or β-sheet, and loops that connect the helices and strands. Some studies suggest that certain residues appear more often in helices than in strands, which implies a correlation between amino-acid sequence and the shape. It seems that this correlation is not completely understood. An aggregate of all these localized secondary structures forms the *tertiary structure* which is really important in a protein's function. Tertiary structures can also be combined as subunits to form a larger *quaternary structure*. For the RNA case the secondary structures are the main interest. In fact the way we approach the problem does not depend on the structure level since we can view each sub-unit as a primary structure and recursively apply the same models on those structures.

Protein conformation and stability is influenced by a number of factors that include hydrogen bonds and hydrophobic effects (Socci et al., 1994). The amino acid chain transforms to very distinctive three dimensional structures because of those factors and this process is called *protein folding*. The function of a protein is tied to its structure, so being able to quickly specify a structure from its amino acid sequence is of significant interest.

Unfortunately, finding those structures remains elusive due to the astronomical number of possible conformations. The complexity can be found by comparing another problem, finding the lowest energy conformation of simple atomic clusters in which the entities are far less complex. This simpler problem is shown to be NP-hard by Greenwood (1999). X-ray crystallography (XC) and nuclear magnetic resonance (NMR) are two methods that can be used to determine the structure, but both methods are time consuming; only a small percentage of proteins have been studied using those methods.

In this section we will summarize three different sub-problems in protein folding and their solutions by evolutionary algorithms.

3.2.1. Minimalist Models

This approach tries to predict the fold without using the structure information from any other protein for comparison. They try to explore an energy hyper surface for a minimal energy conformation, which is believed to correspond to the *native state*. Here again, the hyper surface is very big and the search process is complicated. In order to reduce this effect the researchers are using minimalist models where the shape of the chain is restricted to form a self-avoiding walk on a square lattice (Figure 6).

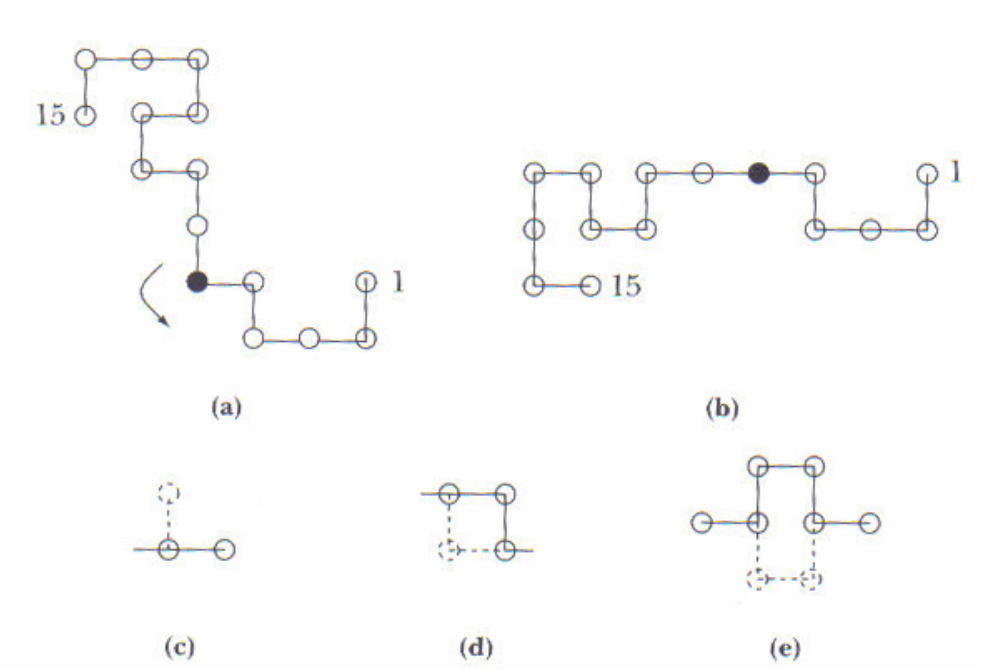


Figure 6 : A conformation of a chain on the two-dimensional lattice. (a-b) an example of a $+90^{o}$ rotation mutation. Note -90^{o} is not self-avoiding. Other simple mutations are shown in (c-e). (Fogel and Corne 2003)

Konig and Dandekar (1999) described a genetic algorithm that uses a systematic crossover operator to search for low-energy conformations of two dimensional primitive models. The operator begins with choosing a parent with a biased probability and then tests each possible crossover point, the two best individuals are selected for the next generation.

The diversity in the population is also checked in this GA framework. After every ten generations, newly created individuals are tested to see if they differ from every individual of the parents' population and discarded if not. Fitness was measured by a simple energy function: Add -1 for each pair of unconnected *hydrophobic residues* that reside at non-diagonal neighboring lattice points. For example, if residues 10-15 in Figure 6 are hydrophobic, this conformation would have an energy of -2 because of the interaction of residue pairs (10,13) and (10,15). The resultant structure will have those hydrophobic residues in the interior side since the lowest energy conformation will have a core in the middle formed by hydrophobic residues (Figure 7). This approach can be generalized to a three-dimensional cubic lattice that can predict secondary structures since with the two-dimensional one we can only predict the hydrophobic core.

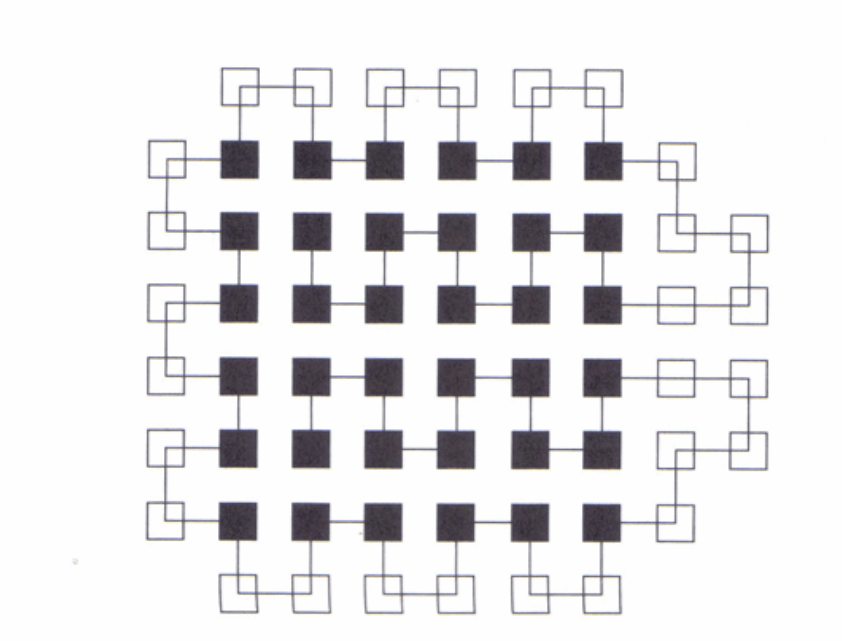


Figure 7 : Lowest energy state of a 64-residue chain. Black blocks are hydrophobic residues. Konig and Dandekar (1999)

Gunn (1997) has used a genetic algorithm with off-lattice models , in which the residues are not required to occupy fixed, equally spaced sites and the side chain is modeled using the *dihedral angles* where two angle values specify the relation between the residues. The GA was using an encoding of those angle pairs as bit strings, where those restricted pairs were assigned from a *dihedral library*. The fitness was measured in terms of root mean square deviation from a structure determined by XC or NMR methods.

A more complicated model is introduced by Sun et al. (1999) where they used a GA to predict protein structures in a 210-type lattice model. In this model, each residue is at fixed distance *l* from the next residue in the sequence. The position of a residue from its neighbor in three-dimensional space is restricted to 0, $\pm a$, $\pm 2a$ in each axis. Hence $l = a\sqrt{5}$. This results in 24 possible neighbors for each residue. Actually only two angles θ and Φ, are needed to place a residue relative to the three previous residues in the sequence (Figure 8). The fixed distance between residues and the restricted placement of neighboring residues yield only 11 valid θ values and 30 valid Φ values.

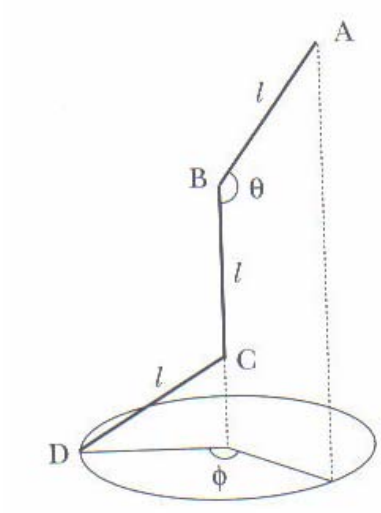


Figure 8: (θ, Φ) pair in a sequence of four residues

The encoding for GA is the angle pairs. The initialization is done randomly. The standard mutation and crossover operators are used to create the offspring. The model predicted the structure of cytochrome protein to within an r.m.s. deviation of 7.5 Å, which is somewhat high, considering that only the backbone was modeled.

3.2.2. Homology Based Modeling

The use of the known structure for a homologous protein can be used to predict the structure of another one. Indeed, as the number of known structures increases, the probability of resolving the conformation of other unsolved proteins will increase.

Wilson et al. (1993) listed the three major aspects in homology-based modeling:

1) Amino acid sequence alignment
2) Generation of loop conformations when necessary
3) Side chain conformation prediction

The side-chain conformation problem is the most crucial subproblem that must be solved the whole problem is solved. All proteins are identical if we ignore the side chains. Therefore it is the *side chain packing* that determines feasible conformations.

Despite the extreme importance of side chain packing problem, we could not find much interest from Evolutionary Computation community to solve this problem. Most homology modeling using genetic algorithms are used for sequence alignments (Notredame et al., 1998). However Desjarlais and Handel (1995) used a GA to search for low-energy hydrophobic core sequences and structures. In their approach, each core position was allocated a set of bits within a binary string, and the bit values encoded a specific residue type and set of torsion angles as specified in the library. Here the torsion angles mean the rotation angle between the side chain and the C_α carbon. Therefore the input is a list of residue/torsion possibilities for the string location corresponding to the core position. In the GA, reproduction is performed by using standard one point crossover and bit flip mutations. An inversion operator was added to establish genetic linkage between pairs of bits. This GA was later used by Ghirlanda et al. (1998) in a small peptide chain modeling in which an evolution strategy (ES) is used.

Ghirlanda et al. constructed an ES predictor that used the polypeptide model with all backbone atoms represented including C_β atom (a carbon atom inside side chain directly bonded with C_α atom). In this model the energy is measured by the r.m.s deviation from a known crystal structure.

The genotype in this ES predictor is an integer array, which describes the torsion angles of each residue. The survival of the individuals is determined by $(\mu+\lambda_t)$-ES strategy where at generation *t,* μ parents produce λ_t offspring and parents compete as equals with offspring survival. Reproduction on the other hand is handled differently. Each parent may generate up to 200 offspring, however decision on the low-fit offspring is done immediately. This results with parents having different number of offspring. Here λ_t and λ_s are not necessarily equal if $t \neq s$. That's how it differs from the conventional $(\mu+\lambda)$-ES strategy. After reproduction, the truncation selection chooses μ parents for the next generation.

New structures in the ES are generated by stochastic modification of selected residue torsion angles. Mutation is the only reproduction operator because in their research they have noticed that recombination has less impact since the structures are more compact. Mutation is performed over randomly positioned set of *k* consecutive residues (i.e k=3). The r.m.s deviation from the crystal structure, which defines the fitness function, is computed only over a window of consecutive residues including the ones that are mutated. The window size *w* is a parameter that was expanded until it equals the full length of the polypeptide chain.

They have tested the ES predictor with a protein which has 61 residues. This protein contains one α-helix and four β-strands. Each generation manipulated a population size μ=200 with λ_t = 200 (maximum value). The initial window size was *w*=5. The predictor consistently found conformation with a r.m.s deviation of approximately 1.8 Å. Normally it is said the prediction is "correct" if the deviation is less than 1 Å in which Ghirlanda et al. believe with a final relaxation they can reduce the deviation under that level.

3.2.3. Docking Models

In this approach the researchers are trying to predict how two organic molecules will energetically and physically bind together. One molecule, called *receptor* contains "pockets" that form binding sites for the second molecule, which is called *ligand*. Any solution must therefore describe both the shape of the receptor and the ligand, as well as their affinity. The importance of this approach can be understood by the drug design approach. Drug designers attempt to determine the particular drug that best binds to a protein pocket. Here if we try to do an exhaustive search it will be impossible to solve the problem. For example AIDS virus depends on the HIV protease enzyme. If one could find a small molecule that would permanently bind to the active site in the protease, the function of that enzyme would be prevented.

Docking approach is used by Morris et al. (1998) in conjunction with an elitist GA. They used the software package AutoDock 3.0 to attain the energy composition.

The genotype was composed of a string of real-valued genes: three Cartesian coordinates for the ligand translation; four variables defining quaternion (a vector defining an

axis of rotation and rotation angle) specifying the ligand orientation, one number for each ligand torsion.

As for genetic operators it seems they have used two point crossover and a mutation operator which adds a Cauchy-distributed random variable to the single gene. Mutation was no longer needed to accomplish local search but was used for jumping in the search space, for exploration. They used both generational and steady-state GA's in their experiments. The fitness was measured by an empirical free energy function:

$$\Delta G = \Delta G_{vdW} + \Delta G_{hbond} + \Delta G_{elec} + \Delta G_{tor} + \Delta G_{sol}$$

where ΔG_{vdW} is the van der Waals dispersion/repulsion energy, ΔG_{hbond} is the hydrogen bonding energy, ΔG_{elec} is the electrostatic energy, ΔG_{tor} is the restriction of internal rotors and global rotation and translation, and ΔG_{sol} models desolvation upon binding and the hydrophobic effects.

In their experiments Morris et al. compared simulated annealing, a generational GA and a steady-state GA where the steady-state GA found the lowest energy and the lowest r.m.s. deviation from the crystal structure.

3.3. Using Parallel Fast Messy Genetic Algorithms

The solution of PFP determines the three-dimensional structures of proteins given only their amino acid sequence. The interest in this problem originates from the Human Genome Project and the huge amount of genetic information gained so far. Currently more than 50,000 proteins are known with their amino acid sequences and with the completion of the Human Genome Project all of them will be known. At that point we will need to have efficient and general protein folding predictors to conform the structures.

Other than predicting the structures of naturally occurring proteins, some researchers are concerned with promoting fast protein design which is sometimes referred as *inverse protein folding problem*. Like in the work of Chan and Dill (1993) and Lengauer (1993), there are applications of pharmaceuticals with fewer side effects, proteins with energy conversion and

storage capabilities (like in photosynthesis). In this section we will refer to those types of polymer of amino acids using the term *polypeptide* and refer to the naturally occurring ones as *protein*.

In this section, we will briefly go over the work done with fast messy genetic algorithms especially the case they are used in identifying three-dimensional structures of arbitrary polypeptides in arbitrary environments.

Genetic algorithms have an underlying principle that says: "Small pieces of solution that exhibit above-average performance can be combined to create larger pieces of above-average quality, which can themselves be recombined into larger pieces, and so forth". This principle is characterized by the *building block hypothesis* (Holland, 1975). This hypothesis is one of the most hotly debated topics in GA literature, and proposed to be the motivation behind messy genetic algorithm (mGA).

Simple GAs suffer from the fact that the pieces that form the building blocks must be put next to each other explicitly in the genotype or else they are more likely to be disrupted by crossover. This problem is magnified when competing schemata (schemata with different values at similar defining positions) define locally optimal solutions. *Deception* occurs when the expected number of copies of locally optimal building blocks is greater than that of globally optimal ones.

Messy GAs are designed to deal with these problems by encoding the string position (locus) along with its value (allele). This gives an mGA the ability to search for the true building blocks of the problem and "create tighter linkage for those genes than a fixed position encoding allows" (Goldberg et al., 1989).

Messy GAs have similar genetic operators as the simple GAs however tournament selection has been used instead of roulette wheel or rank-based selection. This operator also controls the number of positions in common among individuals in order to decide for them to compete. Crossover is replaced by a *cut-and-slice* operator which divides the string and appends those parts to each other to form a longer string. Goldberg does not mention the mutation operator in their mGA implementations. The initialization is done by a complex method (rather than random) with eliminating the useless blocks to keep the above-average building blocks (*primordial phase*).

Fast mGAs (fmGA) are mGA variants designed to reduce the complexity of the initialization phase and thus the overall algorithm time and space complexity (Goldberg et al. 1993).

In their work in U.S Air Force Academy, Michaud et al. (2001) is using a parallelized fmGA to search for the minimum energy conformation of a polypeptide chain (Met-Enkephalin). In their representation there are 24 dihedral-angles represented by 10 bits each. This results with a search-space of $1024^{24} \approx 1.767 * 10^{72}$ conformations. They have stated that even moving to a slightly larger protein such as Polyalinine_{14} results in a search space of nearly $1024^{56} \approx 3.77 * 10^{168}$ conformations.

Applying the fmGA, they seem not to be satisfied with the results with a pure genetic algorithm since they decided to use a gradient-based minimization algorithm in order to perform local search.

They have also conducted experiments to determine the effects of seeding the initial population with three different methods; (1) members that have some secondary structure (α-helix and β-sheet), (2) with locally optimized population members, or (3) with a combination of these two. The experiments included various percentages of the population to be initialized with those methods and using the Met-Enkephalin protein which contains no secondary structure and Polyalinine which has a perfect α-helix secondary structure.

The hybrid seeding (option 3) method converge toward better averages than do the other method tested, but the results are not clear cut. Adding optimized solutions into the initial population often resulted in the algorithm eventually converging to a local optimum. However both the first two methods mentioned allow the fmGA to be the dominating factor in finding better solutions.

3.4. RNA Folding Problem

Similar to PFP, the goal of RNA folding is to fold an RNA sequence into a biologically functional structure that is stable with an optimal or suboptimal free energy. The non-deterministic GA has been adapted to folding RNA sequences and, in addition, has incorporated

the ability to form simple pseudoknots in a natural way. In their GA, Shapiro et al. (2001) create a large population of RNA structures and they distribute those to an extensive number of processors such that each processor holds one RNA structure. All RNA structures are evolved in parallel, one generation at a time through a three-step procedure consisting of the three basic operators: selection, mutation and crossover, using the stems generated from a given RNA sequence. Minimal free energy is used as fitness function to improve the population.

Selection operation is done from the set of nine structures including the structure on the processor itself and the eight-neighbor processors. This choice of parents is made by using a ranked rule biased towards structures with better fitness values.

Figure 9 depicts the efficiency of the algorithm where the average number of generations until convergence of the population seems to increase with population size.

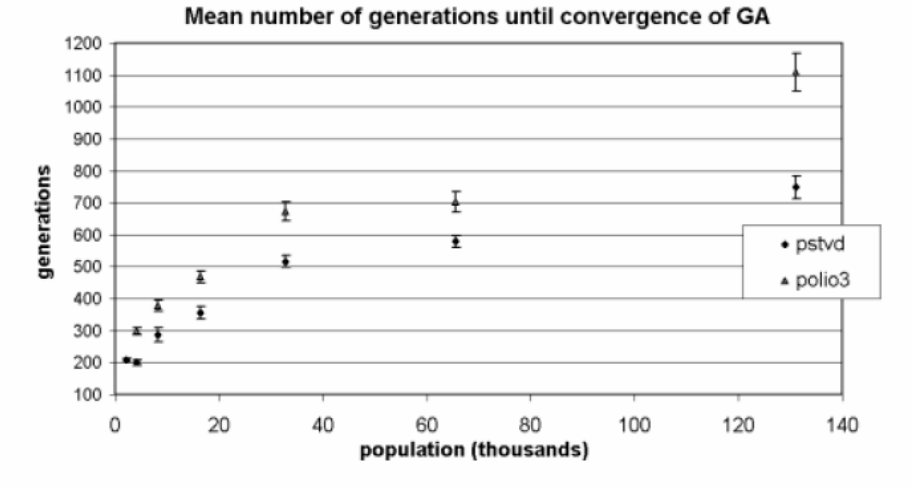


Figure 9: Mean number of generations required until convergence. Mean was calculated over 20 runs. Error bars for data points are shown too. (Shapiro et al., 2001)

Another important aspect of the ability to vary the size of the GA population is the accuracy and fitness of the solutions as a function of that population size. Although it is known that biologically active structural configurations at times do not constitute a global minimum energy structure, biological evidence supports the theory that the change in energy for the formation of the *correct* structure of a particular sequence will usually lie in within the minimum 10% of all possible values of energy (ΔG) for structures compatible with that sequence. In fact, many current approaches to RNA structure prediction are based entirely on determining an

ensemble of possible structures with free energy nearest to the calculated minimum. Thus, it is reasonable to compare thermodynamic fitness of solutions generated at each population when analyzing the performance of the algorithm. Figure 10 indicates that increasing the population of the GA increases the efficacy of the algorithm in locating a highly thermodynamically fit structure.

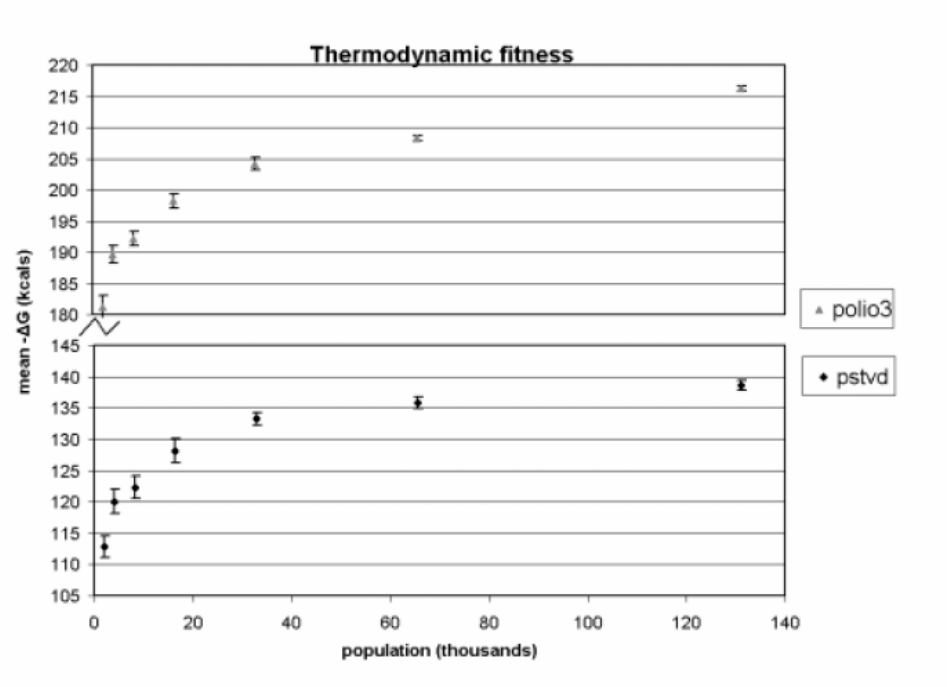


Figure 10: Improvements in thermodynamic stability as a function of population size for fodings of polio 3 and PSTVd. (Shapiro et al., 2001)

RNA folding problem is also approached from another approach where the sequences are modeled with a stochastic context free grammar (SCFG) deriving them. This enables us to analyze the biological properties of RNA, DNA or amino acid sequences in terms of formal language theoretic concepts (Searl, 1993). Our work is also using this approach where we are trying to re-estimate the parameters of the grammar using a steady-state genetic algorithm.

Genetic algorithm estimation for SCFG parameters is proposed by Sakakibara and Kondo (1999) where they are not only focusing the bioinformatics domain but they are investigating more theoretically. However we find it necessary to briefly explain what they did in this section.

Sakakibara and Kondo (1999) propose a genetic algorithm based learning of context-free grammars from a finite sample of positive and negative data. They are indicating the main two problems in grammar learning: determining the grammatical structure (topology) and identifying nonterminals in the grammar. Their method uses a representation which is similar to the table used in the optimum parsing algorithm (i.e Cocke-Younger-Kasami , CYK). By employing this representation method, the problem of learning context-free grammars from examples can be reduced to the partitioning problem of nonterminals. After this step they use genetic algorithms for solving the partitioning problem. Note that there are exponentially many grammars that can generate the given positive example. Thus the hypothesis space of context-free grammars is very large to search a correct context-free grammar consistent with the given examples. Sakakibara, (1992) has also shown that if information on the grammatical structure of the unknown context-free grammar to be learned is available for the learning algorithm, there exists an efficient algorithm for learning the grammar from only positive examples.

They have created a tabular representation of the grammars and used it for the individuals in the genetic algorithms so that each grammar is evaluated with its topology. In Figure 11 the tabular representation of a sequence *w* of length *n* is shown. This triangular table not only defines a grammar that generates *w* but also all possible grammatical structures on *w*.

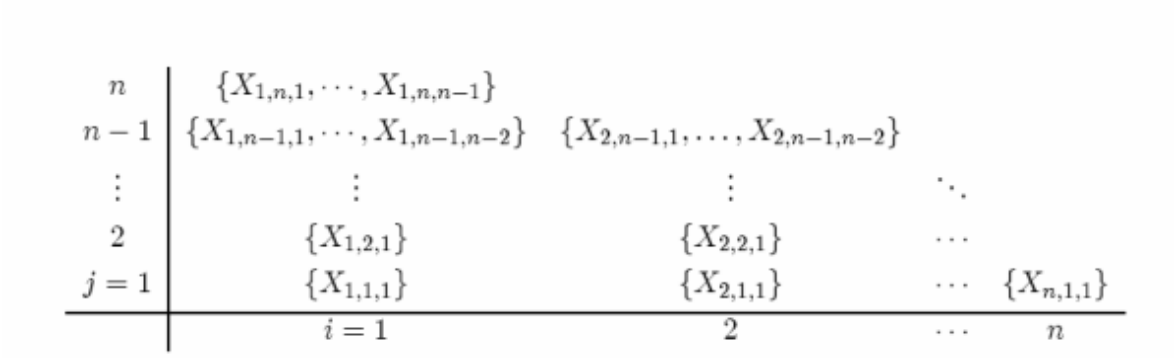


Figure 11 : Tabular representation for a sequence of length *n*.

The algorithm for learning the CFGs is designed as follows: Given positive examples the tabular representation is formed. Distinct nonterminals are merged to be consistent with the given positive and negative examples and minimize the number of nonterminals in the grammar.

Then the GA is used to solve the partitioning problem for the set of nonterminals $\{X_{i,j,k} \mid 1 \le i \le n, 1 \le j \le n-i+1, 1 \le k < j\}$. This problem contains the problem of finding minimum-state finite automata consistent with the given examples and hence it is HP-hard. Therefore, it is reasonable to use the genetic algorithm for solving the computationally hard problem of partitioning nonterminals. The encoding in the GA represents the partitions and the fitness function tries to find a CFG consistent with both of the given positive and negative examples. Especially, the penalty for a CFG which accepts any negative example is huge, it discards all those individual. Among the CFGs consistent with all examples, they follow "Occam's Razor" to choose the best ones which contains minimal number of nonterminals. The paper is also a good reference for future work in theoretical part of grammatical approaches since it is dealing with an efficient representation of grammars.

3.5. Evolved Neural Networks

We found various papers in the literature using artificial neural networks (ANN) for RNA and protein folding problems (Qian N, Sejnowski T J, 1988; Riis S K, Krogh A,1996; Cuff J. A and Barton G.J, 1999). Since we are investigating EA methods, in this section we will focus on neural network models that are optimized and evolved by EA.

However, the evolved neural networks have not been used much in our domain of folding problem. The domain we will consider will be much like a pattern recognition problem, identification of coding regions in DNA sequences.

The DNA sequence information is gained in a very high rate, however they are not being deciphered and annotated as fast. In these sequences there are mainly two regions so-called *coding (exons)* and *non-coding (introns)* regions. Coding regions are known to result in RNA and protein products and identifying them is very significant for biologists. In a newly sequences DNA the major interest is identifying the potential exons which will be referred as *genes* in the genome.

In the literature two general approaches are used. In the first one, the rules or criteria are defined to form exons and the sequences that do not meet specified criteria are eliminated after applying those algorithms such as GeneID (Guigo et al., 1992). In the second approach, an ANN or hidden Markov model (HMM) is used to calculate a set of weighted statistics and determine a composite score, which is used to identify possible exons. These scoring algorithms are used by GRAIL (Uberbacher and Mural, 1991), GRAIL2 (Uberbacher, 1995) and GeneParser (Snyder and Stormo, 1995) frameworks.

The purely rule-based algorithms can generate false negatives simply because ther is insufficient knowledge used in generating the rules. This suggests that a non-rule-based approach for gene identification through machine learning may prove to be a more effective method.

Optimizing ANNs through simulated evolution not only offers a superior search for appropriate network parameters, but the evolution can also be used to adjust the network's topology simultaneously. By mutating both the network topology and its associated weights, a very fast search can be made for a robust design. This approach does not restrict the network with one topology and then trying to search for best weights. Evolving ANNs can be gone over by the paper by Yao (1999). The self-design process is almost automatic; unlike the traditional ANN paradigms that require the active participation of the user as part of the learning algorithm, an evolutionary ANN can adapt to unexpected feature inputs being more robust than the traditional approach and is very capable of machine learning.

Porto et al. (1995) had compared EC with back propagation and simulated annealing in the domain of training a fixed network topology to classify active sonar returns. The results indicated that stochastic search techniques such as annealing and evolution consistently outperform back propagation and they are suitable for parallel processing computers too. The procedure of EA in this approach is efficient because it can use the entire current population of networks as initial solutions to classify each new data. There is no need to restart the search procedure when new data is seen, in contrast with many classic search algorithms such as dynamic programming.

Landavanzo et al., (2002) has demonstrated that EA has generated superior results when used to train ANNs for the DNA exon identification problem. In their framework, a population of complete networks is selected at random. A network is represented by the number of hidden layers, the number of nodes in each of these layers, the weighted connections between

all nodes in a feed-forward or other design, and all the bias terms for each node. There are bounds selected for the size of the network based on the available memory and architecture of the system. The input and output layers are fixed since the input is the sequence and the output is determined by the number of classes to classify. Each of the *parent* networks is evaluated on the sample data. A typical objective function is the mean square error between the target output and the actual output summed over all output nodes. *Offspring* networks are created from those parent networks through random mutation. Simultaneous variation is applied to the number of layers and nodes, and to the values for the associated parameters (e.g. weights and biases of a multilayer perceptron). A probability distribution is used to determine the likelihood of selecting combinations of these variations. This probability distribution can be preselected by the operator or can be made to evolve along with the network, providing for nearly completely autonomous evolution. (Fogel, 2000). Their evolved ANN model is constructed with these parameters which are capable of identifying coding and noncoding regions. It takes a DNA sequence as input and produces a feature table that describes the location and structure of the patterns making up any genes present in the sequence.

The advantages of using ANNs for gene identification include:

1) It is possible to combine different input information about DNA sequences.
2) These various inputs are combined in an unbiased manner.
3) The system may be robust to input noise (sequencing errors) because of the redundancy in information and partially independent nature of the input data.

The research presented here suggests that the best-evolved artificial neural network performed better than GRAIL considering guanine and cytosine percentage. In some cases, the evolved ANN performed at a sensitivity double that of GRAIL. In terms of exons overlapped, on average the evolved ANN was capable of outperforming GRAIL when the sequences were biased toward a high guanine and cytosine percentage.

The training set used for the ANN consisted of an equal number of coding and noncoding nucleotides. In reality, it has been estimated that only 2% of human DNA is coding and the remainder is noncoding. Use of an equal proportion of the two classes for training may have biased the evolved ANN to become artificially sensitive to coding data and less capable of classifying noncoding data. As a result, it classified the majority of coding nucleotides correctly

(true positives), and often classified noncoding nucleotides incorrectly as coding (false positives). This approach led to a conservative estimate of coding regions with a low probability of completely missing a coding region. This will also lead the user of the ANN to omit the mislabeled exons entirely from further analysis. When the algorithm does predict an exon, it will have a high probability of correct classification.

3.6. Concluding Remarks

The EA applications used in secondary structure prediction have resulted in the development of sound algorithms for finding the optimal conformations of a given sequence. However, there are still many difficulties inherent in finding these optimal conformations for large proteins because of limitations on computational speed and the high dimensionality of the problem. The fmGAs have proved to be an interesting and effective technique. Combining this with other methods should provide more effective and efficient approaches than we currently have for predicting secondary structure. For example, the incorporation of secondary structure or peptide backbone information into elements of various genetic algorithms may yield improved results for proteins. Moreover, enabling EAs to conduct additional localized searches generally results in a significant improvement in the fitness of the best solution found, especially with steady-state approaches.

Note that similar EA techniques involving energy minimization can also be applied to the protein sequencing, protein docking and RNA folding problems. However in RNA folding problem, the encoding will be different from dihedral or torsion angles, instead we will be using the hydrogen bonds and pairing energies in order to conduct the search.

After we have gone through all these EA techniques we would like to suggest some directions in research that can be pursued in hopes of identifying conformations and secondary structures in a more effective manner.

1) Integrate manual adjustments for a biologist skilled at pattern recognition. The application would let him or her to move the evolving solution away from a local optimal or to further explore a promising part of the search space.
2) Change the termination criteria so that EAs execute until no better energy minimized solution is found. Consequently, a subpopulation can be randomized and added to the current population and the search process can be continued.
3) Use real values instead of binary values (e.g. in our work) in order to prevent the disruption of building blocks.
4) Group the various energy functions according to their models for interactions of all atoms in a sequence. After this, a *multi-objective* EA can be applied to the problem comparing different groups of energy functions. These multi-objective fitness functions would allow the EA to be expanded to perform additional tasks such as detecting the secondary structure.
5) Find the optimal EA parameter settings to obtain better results for different sequences. Use these settings when determining building block size in fmGA. For ES and EP approaches, try various mutation operator parameters, population size, the use of $(\mu+\lambda)$ versus (μ,λ), etc.
6) Add pattern recognition algorithms (such as neural networks) for determining secondary structure based on known folding structures and their locations with respect to the protein chain (homology). Current techniques used to identify secondary structure a priori have resulted with nearly 75% success.
7) If there are several subfunctions for fitness function, use only a few of them in initial search; as the search proceeds, add more subfunctions to the evaluation process. On the other hand using not adequate fitness functions can result with exploration in unpromising parts of the search space.
8) Use parallel processing both for population calculations and the energy-fitness function models.
9) Use GAs to adjust the topology and the weights of an artificial neural network which can be used for any kind of prediction problem such as DNA coding region identification, RNA classification, phylogeny problems etc.

10) Consider the grammatical approach in modeling sequences since they are more powerful in database search and homology prediction. GAs can be used to find the topology of the grammar or parameter estimation like in our work.

4. THE GENETIC ALGORITHM MODEL

4.1. Genetic Algorithm Components

Genetic Algorithms are a family of computational models inspired by evolution. These algorithms encode a potential solution to a specific problem on a simple chromosome-like data structure and apply recombination operators to these structures so as to preserve critical information. Genetic algorithms are often viewed as function optimizers although the range of problems to which genetic algorithms have been applied is quite broad.

An implementation of a genetic algorithm begins with a population of (typically random) chromosomes. One then evaluates these structures and allocates reproductive opportunities in such a way that those chromosomes which represent a better solution to the target problem are given more chances to "reproduce" than those chromosomes which are poorer solutions. The "goodness" of a solution is typically defined with respect to the current population.

This particular description of a genetic algorithm is intentionally abstract because in some sense the term genetic algorithm has two meanings. In a strict interpretation, genetic algorithm refers to a model introduced and investigated by John Holland (Holland, 1975). It is still the case that most of the existing theory for genetic algorithms applies either solely or primarily to the model introduced by Holland as well as variations on what are referred to as the *canonical genetic algorithm*. Recent theoretical advances in modeling genetic algorithms also apply primarily to the canonical genetic algorithm.

In a broader usage of the term, a genetic algorithm is any population-based model that uses selection and recombination operators to generate new sample points in a search space. Many genetic algorithm models have been introduced by researchers largely working from an experimental perspective. Many of these researchers are application oriented and are typically interested in genetic algorithms as optimization tools.

Usually only two main components of most genetic algorithms are problem dependent: the problem encoding and the evaluation function.

Consider a parameter optimization problem where we must optimize a set of variables either to maximize some objective function such as profit or to minimize cost or some measure of error. We might view such a problem as a black box with a series of control dials representing different parameters; the only output of the black box is a value returned by an evaluation function indicating how well a particular combination of parameter settings solves the optimization problem. The goal is to set the parameters $(X_1, X_2, ..., X_M)$ so as to optimize (minimize or maximize) some objective function $F(X_1, X_2, ..., X_M)$.

Most users of genetic algorithms typically are concerned with problems that are nonlinear. This also often implies that it is not possible to treat each parameter as an independent variable which can be solved in isolation from the other variables. There are interactions such that the combined effects of the parameters must be considered in order to maximize or minimize the output of the black box. In the genetic algorithm community the interaction between variables is sometimes referred to as *epistasis*.

The first step in the implementation of any genetic algorithm is to generate an initial population. In the canonical genetic algorithm each member of this population will be a binary string of length *L* which corresponds to the problem encoding. Each string is sometimes referred to as a "*genotype*" (Holland, 1975) or alternatively, a "*chromosome*" (Schaffer, 1987). In most cases the initial population is generated randomly. After creating an initial population, each string is then evaluated and assigned a "*fitness*" value.

The notion of evaluation and fitness are sometimes used interchangeably. However, it is useful to distinguish between the evaluation function and the fitness function used by a genetic algorithm. The evaluation function or objective function provides a measure of performance with respect to a particular set of parameters. The fitness function transforms that measure of performance into an allocation of reproductive opportunities. The evaluation of a string representing a set of parameters is independent of the evaluation of any other string. The fitness of that string, however, is always defined with respect to other members of the current population.

In the canonical genetic algorithm, fitness is defined by f_i / f where f_i is the evaluation associated with string *i* and *f* is the average evaluation of all the strings in the population. Fitness

can also be assigned based on a string's rank in the population (Baker, 1985; Whitley, 1989) or by sampling methods, such as tournament selection (Goldberg, 1990).

It is helpful to view the execution of the genetic algorithm as a two stage process. It starts with the *current population*. Selection is applied to the current population to create an *intermediate population*. Then recombination and mutation are applied to the intermediate population to create the *next population*. The process of going from the current population to the next population constitutes one generation in the execution of a genetic algorithm (Figure 12). Goldberg (1989) refers to this basic implementation as a Simple Genetic Algorithm (SGA).

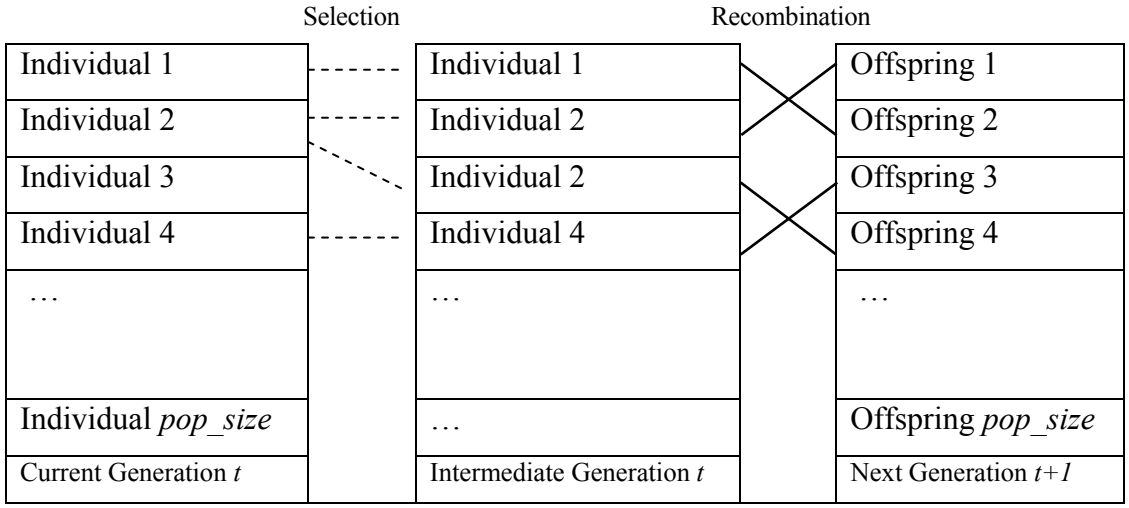


Figure 12 : The generation process in a typical GA iteration

4.2. Encoding

The encoding of a possible solution is called *genotype*. In our proposed system the genotype consists of the probability parameters associated with the rules in the grammar. The floating point numbers from the range [0,1] are used. The length of the genotype depends on the number of rules in the given grammar structure. The *phenotype* is a possible solution in a given genotype. A possible genotype in our system can be the parameter set indicated inside curly braces for the following SCFG:

```
<S>   : <baseA> <S-and-baseA> {0.054261}
      | <baseA> <S-and-baseC> {0.054726}
      | <baseA> <S-and-baseG> {0.031067}
      | <baseA> <S-and-baseU> {0.025713}
      | <baseC> <S-and-baseA> {0.051464}
      | <baseC> <S-and-baseC> {0.048842}
      | <baseC> <S-and-baseG> {0.032937}
      | <baseC> <S-and-baseU> {0.043907}
      | <baseG> <S-and-baseA> {0.043119}
      | <baseG> <S-and-baseC> {0.009975}
      | <baseG> <S-and-baseG> {0.033629}
      | <baseG> <S-and-baseU> {0.064802}
      | <baseU> <S-and-baseA> {0.056856}
      | <baseU> <S-and-baseC> {0.074478}
      | <baseU> <S-and-baseG> {0.098345}
      | <baseU> <S-and-baseU> {0.030334}
      | <base> <S> {0.009047}
      | <S> <base> {0.003038}
      | <S> <S> {0.046698}
      | A {0.072100}
      | C {0.081618}
      | G {0.017935}
      | U {0.015111};
<S-and-baseA>     : <S> <baseA> {1.000000};
<S-and-baseC>     : <S> <baseC> {1.000000};
<S-and-baseG>     : <S> <baseG> {1.000000};
<S-and-baseU>     : <S> <baseU> {1.000000};
<baseA>     : A {1.000000};
<baseC>     : C {1.000000};
<baseG>     : G {1.000000};
<baseU>     : U {1.000000};
<base>      : A {0.194008}
            | C {0.226079}
            | G {0.230061}
            | U {0.349853};
```

The grammar rules need to follow the property that the sum of probabilities must be 1 for the rules that have the same nonterminal on the left hand side. The phenotype needs to be repaired after every mutation or recombination process to conform this property. This procedure adds to our program extra linear time in the number of rules. The procedure can be explained as:

$$P'(X \to YZ) = \frac{P(X \to YZ)}{\sum_{Y,Z} P(X \to YZ)}$$

where $P'(X \to YZ)$ is the repaired value for the probability associated with the rule $X \to YZ$. The procedure simply normalizes the parameter according to the sum of the probabilities associated with the rules with the same left-hand-side nonterminal.

4.3. Fitness Function

The fitness function is a way to describe the dynamics of genotype frequencies in populations of reproducing individuals. The fitness function measures the (potential for) reproductive success of any individual in a given environment. In our system the fitness function measures the likelihood of the grammar parameters to generate some given sequences. Here the system can be adjusted to a set of sequences by simply adding the likelihood values together in order to achieve a more general result. By using this approach the parameters can be adjusted to a general structure where we can predict unseen secondary structures with the same grammar. This approach can also be seen as *profiling* where we try not only to find the structure of a sequence but also construct a generic structure template for a family of RNA sequences.

We have chosen the Inside Algorithm described in Section 2.4, to evaluate an individual since this algorithm gives the total probability to generate the sequence with the given parameter set. The values coming from this evaluation function are then transformed to fitness values. Here we are using *linear scaling* which creates fitness values for each individual in a linear proportion of their evaluation function results, the returning value from the Inside Algorithm.

In building up our system we have used several fitness function schemes slightly different from each other. Initially we have used the CYK algorithm that finds the optimal probability to derive a sequence from a given grammar. This algorithm is also a dynamic programming approach similar to Inside Algorithm, however it is finding the maximum probability among different derivation trees instead of summing them (Section 2.5). The results we get from this approach are not reliable enough since the parameter set is not guaranteed to conform to the other sequences sharing similar structures (structural homology). GA can converge to a parameter set where the CYK result would be optimum but the model would not fit to other sequences. In addition the parameter set could be adjusted in a way that the non-canonical base pairs have higher probabilities to attain a high derivation probability although it would not be a realistic model conflicting with the biological rules creating RNA sequences. Therefore Inside Algorithm is working better for the objective function.

Next, we have used single sequence to train the GA. In this approach the system has converged to parameter sets satisfying the training sequence. Homology search on the other hand would give biased results since the output grammar is only evaluated on one sequence.

Finally we have decided to use a set of sequences evaluated by Inside Algorithm. The fitness value is calculated by adding up the output values coming from the objective function executed on each sequence.

4.4. Selection and Replacement Methods

Selection is a genetic operator that chooses a chromosome from the current generation's population for inclusion in the next generation's population. Before making it into the next generation's population, selected chromosomes may undergo crossover and / or mutation (depending upon the probability of crossover and mutation) in which case the offspring chromosome(s) are actually the ones that make it into the next generation's population. Currently we are using Roulette Wheel Selection. This is a selection operator for which the chance of a chromosome getting selected is proportional to its fitness (or rank). This is where the concept of survival of the fittest comes into play.

Use of this method nevertheless gives rise to two types of problems: A *super-individual* being too often selected, the whole population tends to converge towards its genome. The diversity of the genetic pool becomes too reduced to allow the genetic algorithm to progress. In addition, with the progression of the genetic algorithm, the differences between fitness are reduced. The best ones then get quite the same selection probability as the others and the genetic algorithm stops progressing.

In order to solve these problems, it's necessary to transform the fitness values using suitable *scaling* schemes. Apply a linear transformation to each fitness (i.e. $f' = a.f + b$) is the most common scaling technique. If the fitness scores can have negative values, then linear scaling would not work correctly, this is the case in our system. To overcome this problem we have used the trivial technique- adding a constant value to the fitness values in order to convert them positive since all of the fitness values are originally negative. Since this constant value can

not be determined in some cases, we have also used a more sophisticated scaling suitable for negative fitness values which is called Sigma Truncation. This method scales based on the variation from the population average and truncates arbitrarily at 0. The mapping from objective to fitness score for each individual is given by the equation $f' = f - (\hat{f} - c.\sigma)$ where $\hat{f}$ is the average fitness, σ is the population standard deviation and c is a reasonable multiple of sigma (usually $1 \leq c \leq 3$).

Since we are using Steady State Genetic Algorithms schema in our system, replacement strategy also plays an important role. A constant percentage of the population is replaced with the offspring while others are copied directly to the next population. The worst individuals are replaced with the given replacement percentage.

4.5. Genetic Operators

The function of genetic operators is to cause chromosomes created during reproduction to differ from those of their parents. They must be able to create configurations of genes that were never existent before and that are likely to perform well.

When genetic operators are used with reproduction plans, the result is a surprisingly sophisticated set of adaptive plans. The two most commonly used genetic operators are mutation and crossover. Briefly mutation operator causes movements in the search space, in addition it restores the lost data. In some EAs especially the case of Evolutionary Strategies (ES) mutation is the only genetic operator, where it modifies a solution with random some changes in the phenotype. Crossover on the other hand aims to combine the above-average building blocks and tries to form the schemata that lead to the optimal solution.

4.5.1. Heuristic Crossover

The heuristic crossover (Wright, 1991) is a recombination operator that assumes the better solution vector exists in the direction of the better parent vector. The movement in the search space has a bias towards the better parent. We are choosing a random number from a uniform distribution such as $p \sim U(0,1)$. Using two parameter vectors, x and y, we compute $z = p.(x - y) + x$. If z satisfies all constraints, we use it as an offspring. Otherwise we choose another p value and repeat this process until a preset maximum number of trials has been reached. If there are no feasible individuals coming out of the crossover, we set z equal to the better of x and y. In this fashion we produce two z vectors namely two children out of two parents.

4.5.2. Gaussian Mutation

Gaussian mutation adds a random value from a Gaussian distribution to each element of an individual's vector to create a new offspring. The random number from the probability density function shown below is chosen.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$$

where μ is the location parameter and σ is the scale parameter.

The random number from [-1,1] is added to the current value of the gene which is chosen by the mutation probability preset by the program. The Gaussian mutation tends not to change the values drastically so that we aim to preserve the gene distribution in most cases to avoid destructive effect of mutation. The mutation probability is decreased gradually as the generations go since the solutions are expected to be converging to optima and we do not want to jump to other areas in the search space.

5. EMPIRICAL AND THEORETICAL ANALYSIS

5.1. Simple Sequence Analysis

In our experimentation we have started with artificially generated or parts of real sequences with simple secondary structures. We wanted to test the system with two different grammar models, a general one and a more specific one. The results are better with a specific grammar describing the secondary structure with more restrictions. The population replacement parameter is set to 0.6, mutation probability is 0.005 and the crossover probability is 0.9 if not indicated otherwise. These parameters are set to the most commonly used values. Since we are using heuristic crossover as the only operator, we are compensating the greediness of the recombination by applying a relatively small replacement rate.

The procedure of the experiments can be summarized as:

1) Create a sequence set for training
2) Create a grammar topology
3) Estimate SCFG parameters with GA model
4) Predict the optimum secondary structure using the output grammar by CYK algorithm
5) Analyze time, accuracy and derivation probability

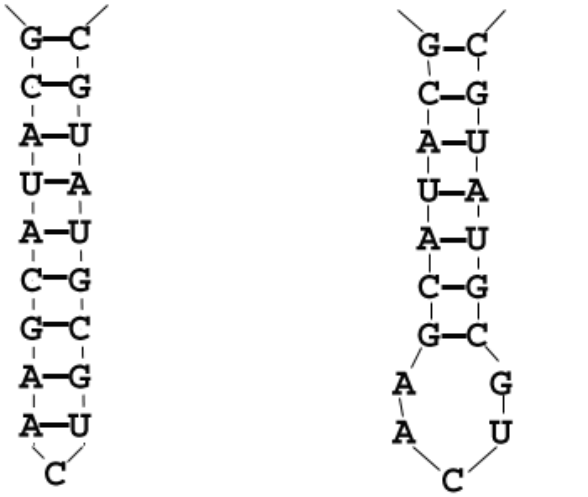
5.1.1. Basic Structure (one stem)

Below output is from the case where the grammar in general allows every kind of secondary structure with parallel and nested stems. The output grammar is given as input to the CYK optimum derivation prediction and the stem of the sequence is predicted to be longer than expected (Figure 13). When the parameters are observed carefully, the rules which represent the

Watson-Crick pairs have higher probability than the other rules which is an expected result. In this experiment there is only one training sequence used as training. Note that the more sequences we have, the more accurate prediction we will get for the unknown data. The incorrect prediction results from the fact that training does not distinguish any pair, any stem or loop structure. The example is trying to estimate parameters of a grammar that has 35 rules and thus has an encoding scheme of 35 alleles. Looking more closely to the RNA sequence of length 19 (*w1*), the original annotation has a stem of length 7 and a loop of length 5 (Figure 14). After GA found the best grammar fitting the sequence, the stem is predicted to be longer by the CYK algorithm. Notice that there is another Watson-Crick pair (A-U) inside the loop and the output grammar has combined it with the existing stem. This result is inevitable considering the fitness of this grammar since it gives the optimum parameter set in which the case extending the stem with a payoff for a non-canonical pair.

```
rna gcauacgaacugcguaugc
Popsize:50   Ngenerations:50   replacement:0.600000   mutation   prob:0.005000
cross.prob:0.900000
FIRST TYPE
<S>    |a {0.042998}
<S>    |c {0.059121}
<S>    |g {0.000344}
<S>    |u {0.017056}
<baseA>      |a {1.000000}
<baseC>      |c {1.000000}
<baseG>      |g {1.000000}
<baseU>      |u {1.000000}
<base>       |a {0.485984}
<base>       |c {0.179399}
<base>       |g {0.128463}
<base>       |u {0.206154}
SECOND TYPE
<S>    |<baseA> <S-and-baseA> {0.011685}
<S>    |<baseA> <S-and-baseC> {0.067844}
<S>    |<baseA> <S-and-baseG> {0.056858}
<S>    |<baseA> <S-and-baseU> {0.083651}
<S>    |<baseC> <S-and-baseA> {
0.039075}
<S>    |<baseC> <S-and-baseC> {0.009973}
<S>    |<baseC> <S-and-baseG> {0.071658}
<S>    |<baseC> <S-and-baseU> {0.002127}
<S>    |<baseG> <S-and-baseA> {0.025782}
<S>    |<baseG> <S-and-baseC> {0.087323}
<S>    |<baseG> <S-and-baseG> {0.021911}
<S>    |<baseG> <S-and-baseU> {0.055544}
<S>    |<baseU> <S-and-baseA> {0.090648}
<S>    |<baseU> <S-and-baseC> {0.064194}
<S>    |<baseU> <S-and-baseG> {0.071331}
<S>    |<baseU> <S-and-baseU> {0.046196}
<S>    |<base> <S> {0.025719}
<S>    |<S> <base> {0.008573}
<S>    |<S> <S> {0.040389}
<S-and-baseA>      |<S> <baseA> {1.000000}
<S-and-baseC>      |<S> <baseC> {1.000000}
<S-and-baseG>      |<S> <baseG> {1.000000}
<S-and-baseU>      |<S> <baseU> {1.000000}

The annotation output from CYK:

seq:gcauacgaacugcguaugc
out:aaaaaaaaa_AAAAAAAAA
org:aaaaaaa_____AAAAAAA

CYK prob in log is: -25.687428

It took 5.658 seconds to run the GA
```

Figure 13 : Output for simple sequence *w1* with a general grammar.



a) predicted structure b) original structure

Figure 14: Comparison of the original and predicted output when a generic grammar model is used.

In Figure 15 the output using a more specific grammar for the same sequence (*w1*) is shown. The grammar is constructed to permit only this kind of structure and the accuracy with this grammar topology is 100%.

```
rna gcauacgaacugcguaugc
Popsize:100 Ngenerations:100 replacement:0.600000 mutation prob:0.005000 cross.prob:0.900000
FIRST TYPE
<loop.Cc>    | a {0.000000}
<loop.Cc>    | c {0.203777}
<loop.Cc>    | g {0.063131}
<loop.Cc>    | u {0.201947}
<baseA>    | a {1.000000}
<baseC>    | c {1.000000}
<baseG>    | g {1.000000}
<baseU>    | u {1.000000}
SECOND TYPE
<stem.Cc>    |<baseA> <stem.Cc.baseA> {0.003487}
<stem.Cc>    |<baseA> <stem.Cc.baseC> {0.034568}
<stem.Cc>    |<baseA> <stem.Cc.baseG> {0.048881}
<stem.Cc>    |<baseA> <stem.Cc.baseU> {0.048618}
<stem.Cc>    |<baseC> <stem.Cc.baseA> {0.059904}
<stem.Cc>    |<baseC> <stem.Cc.baseC> {0.033445}
<stem.Cc>    |<baseC> <stem.Cc.baseG> {0.054872}
<stem.Cc>    |<baseC> <stem.Cc.baseU> {0.026890}
<stem.Cc>    |<baseG> <stem.Cc.baseA> {0.003195}
<stem.Cc>    |<baseG> <stem.Cc.baseC> {0.079194}
<stem.Cc>    |<baseG> <stem.Cc.baseG> {0.032509}
<stem.Cc>    |<baseG> <stem.Cc.baseU> {0.047250}
<stem.Cc>    |<baseU> <stem.Cc.baseA> {0.045543}
<stem.Cc>    |<baseU> <stem.Cc.baseC> {0.017097}
<stem.Cc>    |<baseU> <stem.Cc.baseG> {0.001664}
<stem.Cc>    |<baseU> <stem.Cc.baseU> {0.056553}
<stem.Cc>    |<baseA> <loop.Cc.baseA> {0.042673}
<stem.Cc>    |<baseA> <loop.Cc.baseC> {0.020902}
<stem.Cc>    |<baseA> <loop.Cc.baseG> {0.001493}
<stem.Cc>    |<baseA> <loop.Cc.baseU> {0.047391}
<stem.Cc>    |<baseC> <loop.Cc.baseA> {0.024023}
<stem.Cc>    |<baseC> <loop.Cc.baseC> {0.052233}
<stem.Cc>    |<baseC> <loop.Cc.baseG> {0.050515}
<stem.Cc>    |<baseC> <loop.Cc.baseU> {0.034715}
<stem.Cc>    |<baseG> <loop.Cc.baseA> {0.017499}
<stem.Cc>    |<baseG> <loop.Cc.baseC> {0.022419}
<stem.Cc>    |<baseG> <loop.Cc.baseG> {0.035418}
<stem.Cc>    |<baseG> <loop.Cc.baseU> {0.002397}
<stem.Cc>    |<baseU> <loop.Cc.baseA> {0.000000}
<stem.Cc>    |<baseU> <loop.Cc.baseC> {0.019674}
<stem.Cc>    |<baseU> <loop.Cc.baseG> {0.004383}
<stem.Cc>    |<baseU> <loop.Cc.baseU> {0.030594}
<stem.Cc.baseA>    |<stem.Cc> <baseA> {1.000000}
<stem.Cc.baseC>    |<stem.Cc> <baseC> {1.000000}
<stem.Cc.baseG>    |<stem.Cc> <baseG> {1.000000}
<stem.Cc.baseU>    |<stem.Cc> <baseU> {1.000000}
<loop.Cc.baseA>    |<loop.Cc> <baseA> {1.000000}
<loop.Cc.baseC>    |<loop.Cc> <baseC> {1.000000}
<loop.Cc.baseG>    |<loop.Cc> <baseG> {1.000000}
<loop.Cc.baseU>    |<loop.Cc> <baseU> {1.000000}
<loop.Cc>    |<baseA> <loop.Cc> {0.070824}
<loop.Cc>    |<baseC> <loop.Cc> {0.238623}
<loop.Cc>    |<baseG> <loop.Cc> {0.029552}
<loop.Cc>    |<baseU> <loop.Cc> {0.192146}

The annotation output from CYK:

seq:gcauacgaacugcguaugc
out:aaaaaaa_____AAAAAAA
org:aaaaaaa_____AAAAAAA

CYK prob in log is: -27.672251

It took 48.970 seconds to run the GA
```

Figure 15 : Output for simple sequence *w1* with a specific grammar.

In another experiment, our system is tested with another basic stem-loop structured sequence (*w2*) with the grammar allowing this conformation. The grammar has 52 rules this time and the sequence is a partial tRNA sequence having a stem of length 5 and a loop of length 7 this time. The output of the program is indicated in Figure 16. Like the previous example, the conformation is found precisely.

```
rna cucgcuuagcaugcgag
Popsize:100 Ngenerations:60 replacement:0.600000 mutation prob:0.005000 cross.prob:0.900000
FIRST TYPE
<loop.Cc>    | a {0.119045}
<loop.Cc>    | c {0.049180}
<loop.Cc>    | g {0.030301}
<loop.Cc>    | u {0.164389}
<baseA>    | a {1.000000}
<baseC>    | c {1.000000}
<baseG>    | g {1.000000}
<baseU>    | u {1.000000}
SECOND TYPE
<stem.Cc>    |<baseA> <stem.Cc.baseA> {0.052441}
<stem.Cc>    |<baseA> <stem.Cc.baseC> {0.036634}
<stem.Cc>    |<baseA> <stem.Cc.baseG> {0.012800}
<stem.Cc>    |<baseA> <stem.Cc.baseU> {0.020097}
<stem.Cc>    |<baseC> <stem.Cc.baseA> {0.013237}
<stem.Cc>    |<baseC> <stem.Cc.baseC> {0.026532}
<stem.Cc>    |<baseC> <stem.Cc.baseG> {0.308991}
<stem.Cc>    |<baseC> <stem.Cc.baseU> {0.000533}
<stem.Cc>    |<baseG> <stem.Cc.baseA> {0.026582}
<stem.Cc>    |<baseG> <stem.Cc.baseC> {0.032523}
<stem.Cc>    |<baseG> <stem.Cc.baseG> {0.045407}
<stem.Cc>    |<baseG> <stem.Cc.baseU> {0.045417}
<stem.Cc>    |<baseU> <stem.Cc.baseA> {0.016051}
<stem.Cc>    |<baseU> <stem.Cc.baseC> {0.013724}
<stem.Cc>    |<baseU> <stem.Cc.baseG> {0.041506}
<stem.Cc>    |<baseU> <stem.Cc.baseU> {0.019918}
<stem.Cc>    |<baseA> <loop.Cc.baseA> {0.036950}
<stem.Cc>    |<baseA> <loop.Cc.baseC> {0.036839}
<stem.Cc>    |<baseA> <loop.Cc.baseG> {0.011008}
<stem.Cc>    |<baseA> <loop.Cc.baseU> {0.034995}
<stem.Cc>    |<baseC> <loop.Cc.baseA> {0.007229}
<stem.Cc>    |<baseC> <loop.Cc.baseC> {0.000359}
<stem.Cc>    |<baseC> <loop.Cc.baseG> {0.001710}
<stem.Cc>    |<baseC> <loop.Cc.baseU> {0.038845}
<stem.Cc>    |<baseG> <loop.Cc.baseA> {0.008892}
<stem.Cc>    |<baseG> <loop.Cc.baseC> {0.024062}
<stem.Cc>    |<baseG> <loop.Cc.baseG> {0.001601}
<stem.Cc>    |<baseG> <loop.Cc.baseU> {0.011277}
<stem.Cc>    |<baseU> <loop.Cc.baseA> {0.007223}
<stem.Cc>    |<baseU> <loop.Cc.baseC> {0.010024}
<stem.Cc>    |<baseU> <loop.Cc.baseG> {0.012326}
<stem.Cc>    |<baseU> <loop.Cc.baseU> {0.044267}
<stem.Cc.baseA>    |<stem.Cc> <baseA> {1.000000}
<stem.Cc.baseC>    |<stem.Cc> <baseC> {1.000000}
<stem.Cc.baseG>    |<stem.Cc> <baseG> {1.000000}
<stem.Cc.baseU>    |<stem.Cc> <baseU> {1.000000}
<loop.Cc.baseA>    |<loop.Cc> <baseA> {1.000000}
<loop.Cc.baseC>    |<loop.Cc> <baseC> {1.000000}
<loop.Cc.baseG>    |<loop.Cc> <baseG> {1.000000}
<loop.Cc.baseU>    |<loop.Cc> <baseU> {1.000000}
<loop.Cc>    |<baseA> <loop.Cc> {0.148324}
<loop.Cc>    |<baseC> <loop.Cc> {0.191081}
<loop.Cc>    |<baseG> <loop.Cc> {0.114641}
<loop.Cc>    |<baseU> <loop.Cc> {0.183040}

The annotation output from CYK:

seq:cucgcuuagcaugcgag
out:aaaaa_______AAAAA
org:aaaaa_______AAAAA

CYK prob in log is: -23.754328

It took 20.539 seconds to run the GA
```

Figure 16 : Output for simple sequence *w2* with s specific grammar.

The EM estimation for *w2* and the same grammar structure has found the secondary structure correctly but with a lower probability and in longer time. It has shown that our framework can outperform the other approach both by the means of time and accuracy. The summary of this experiment is shown in the chart in Figure 17.

	GA Estimation	EM Estimation
CYK Probability in logarithm	-23.754328	-27.068858
Time spent	20.539	40.173

Figure 17 : Comparison chart for GA and EM

In Figure 18, a part from the derivation output from the CYK traceback is shown. For the sequence and the estimated grammar we can show the derivation tree by our program. In this notation each step of recursion is shown between sections divided by asterisks. The first line a section indicates the rule used for that subsequence, the second line indicates the logarithm of the probability to generate that subsequence and the last line shows the subsequence with its locations in the whole.

```
*************************
<stem.Cc> : <baseC> <stem.Cc.baseG>
Prob: -23.754328
RNA[0..16]cucgcuuagcaugcgag
*************************
<baseC> : c
Prob: 0.000000
RNA[0..0]c
*************************
<stem.Cc.baseG> : <stem.Cc> <baseG>
Prob: -20.780157
RNA[1..16]ucgcuuagcaugcgag
*************************
<stem.Cc> : <baseU> <stem.Cc.baseA>
Prob: -20.780157
RNA[1..15]ucgcuuagcaugcga
*************************
<baseU> : u
Prob: 0.000000
RNA[1..1]u
*************************
...
```

Figure 18 : A part from the output of CYK traceback

5.1.2. Basic Structure (two stems)

We have tested our system with the general grammar again but this time with a sequence which includes two stems. In this type of structure since the grammar has only one nonterminal for generating the stem (i.e <S>), the parameters will have a tendency to converge to the values where the grammar can produce both stems. However if the length of the stems are different the parameters will lead to an average length of a stem that can result with an incorrect prediction of the structure. In the example below, we are using a sequence which has two stems having nearly the same lengths. The output is shown in Figure 19. As expected, the system predicts the structure correctly but with 95% accuracy since the non-canonical pair A-G is added to the stem (Figure 20).

```
rna gcauacgaaguagcaucccggaugcaugcgcguaugc
Popsize:50 Ngenerations:50 replacement:0.600000 mutation prob:0.005000 cross.prob:0.900000
FIRST TYPE
<S>    |a {0.025036}
<S>    |c {0.089702}
<S>    |g {0.028221}
<S>    |u {0.008385}
<baseA>    |a {1.000000}
<baseC>    |c {1.000000}
<baseG>    |g {1.000000}
<baseU>    |u {1.000000}
<base>    |a {0.052783}
<base>    |c {0.033631}
<base>    |g {0.318541}
<base>    |u {0.595045}
SECOND TYPE
<S>    |<baseA> <S-and-baseA> {0.022357}
<S>    |<baseA> <S-and-baseC> {0.032924}
<S>    |<baseA> <S-and-baseG> {0.026184}
<S>    |<baseA> <S-and-baseU> {0.019026}
<S>    |<baseC> <S-and-baseA> {0.023776}
<S>    |<baseC> <S-and-baseC> {0.000109}
<S>    |<baseC> <S-and-baseG> {0.317938}
<S>    |<baseC> <S-and-baseU> {0.020491}
<S>    |<baseG> <S-and-baseA> {0.022534}
<S>    |<baseG> <S-and-baseC> {0.060803}
<S>    |<baseG> <S-and-baseG> {0.063869}
<S>    |<baseG> <S-and-baseU> {0.060868}
<S>    |<baseU> <S-and-baseA> {0.081633}
<S>    |<baseU> <S-and-baseC> {0.012630}
<S>    |<baseU> <S-and-baseG> {0.030641}
<S>    |<baseU> <S-and-baseU> {0.029592}
<S>    |<base> <S> {0.000000}
<S>    |<S> <base> {0.016975}
<S>    |<S> <S> {0.006306}
<S-and-baseA>    |<S> <baseA> {1.000000}
<S-and-baseC>    |<S> <baseC> {1.000000}
<S-and-baseG>    |<S> <baseG> {1.000000}
<S-and-baseU>    |<S> <baseU> {1.000000}

The annotation output from CYK:

seq:gcauacgaaguagcaucccggaugcaugcgcguaugc
out:aaaaaaaa____bbbbbb_BBBBBB____AAAAAAAA
org:aaaaaaa_____bbbbbb_BBBBBB_____AAAAAAA

CYK prob in log is: -50.565884

It took 43.061 seconds to run the GA
```

Figure 19 : Output for a two-stem sequence with a general grammar.

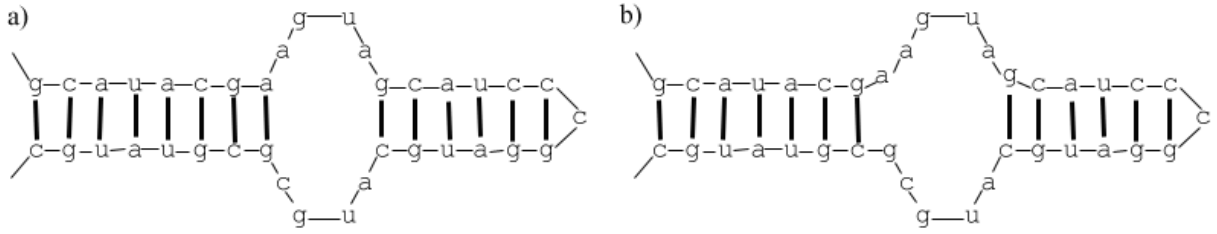


Figure 20 : a) Predicted structure b) Original structure

5.2. Complex Sequence Analysis

In order to test our system with real data we have selected to work on tRNA sequence data, a molecule whose structure has been highly studied and is well known.

tRNA is the information adapter molecule for the organisms. It is the direct interface between amino-acid sequence of a protein and the information in mRNA. Therefore it decodes the information in DNA. There are more than 20 different tRNA molecules known based on the amino acid they carry. All tRNA's from all organisms have a similar structure, indeed a human tRNA can function in yeast cells. Typically there are 4 stems and 3 loops in a tRNA molecule (Figure 21). The part which is called variable loop differs in length in many organisms but in fact it is not a loop structure, just a strand between to stems.

Our GA model is used to predict the structures of the tRNA sequences especially Ala-tRNA from the Homo sapiens genome with a training set of other 7 tRNAs. These molecules are known with their structures found by tRNAscan-SE application (Lowe & Eddy, 1997). tRNAscan is a successful prediction program which had given conforming results with known structures found with X-ray and NMR experiments. We have used a grammar topology with 219 rules which can be applied to any type of tRNA structure. Our program has given 94% accuracy in the structure prediction with population size 100 and 200 generations. The majority of the errors are false positives where the predicted structure has longer stems. This problem results from the fact that non-canonical pairs can have higher probabilities then expected since the contribution of those rules are determined by the rest of the rules having the same nonterminal on the left hand side. If those rules are not used frequently to form the stems or the loops, these non-canonical pairing rules can be rewarded by a small amount of probability. In fact, in a random search algorithm such as GAs this can be an expected result.

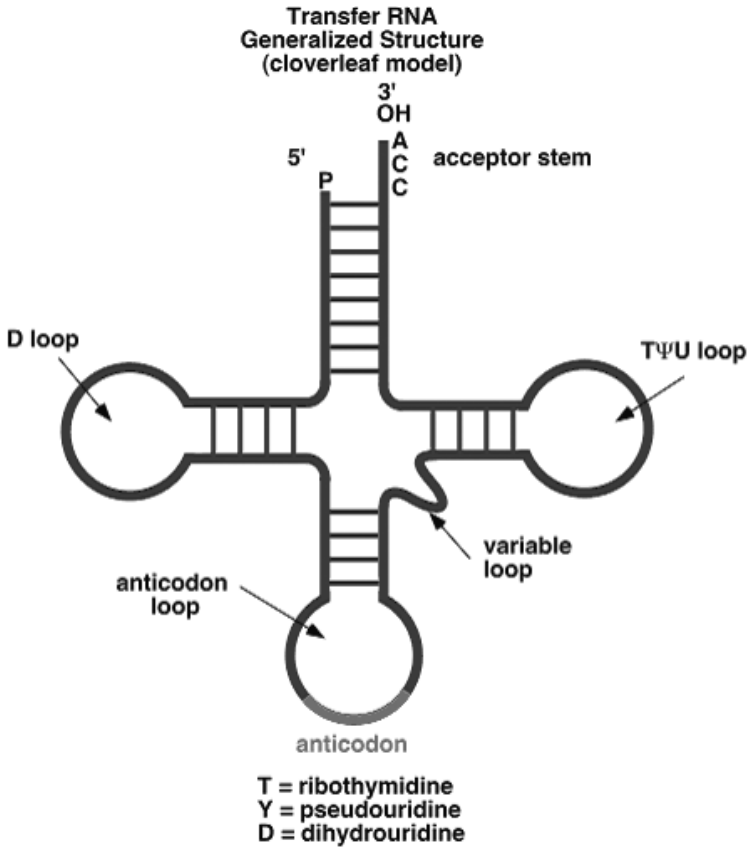


Figure 21 : tRNA general structure

The output annotation is indicated below. In Figure 22 and 23 the derivation trees of the prediction and the correct structure is shown by the output of our program and a diagram respectively. Notice the shifted stem (indicated by B-b) and the longer stem (indicated by D-d) which are the 4 erroneous predicted nucleotides. Here the error comes from the result we have encountered in the simple example too which is estimating the probability of a non-canonical pair more than it should be. Both of the incorrect regions include this pair and in the grammar the reason can be observed with a slightly larger probability assigned to the corresponding rule.

```
seq:ggggaauuagcucaggcgguagagcgcucgcuuagcaugcgagagguagcgggaucgacgcccgcauucucca
out:bbbbbbb_dddd________DDDD__fffff_______FFFFF_____hhhhhh_____HHHHHHBBBBBBB_
org:AAAAAAA..BBBB........bbbb.CCCCC.......ccccc.....DDDDD.......ddddd aaaaaaa.

CYK prob in log is: -108.768784

It took 17343.058 seconds to run the GA
```

Figure 22 : Annotation output for Ala tRNA

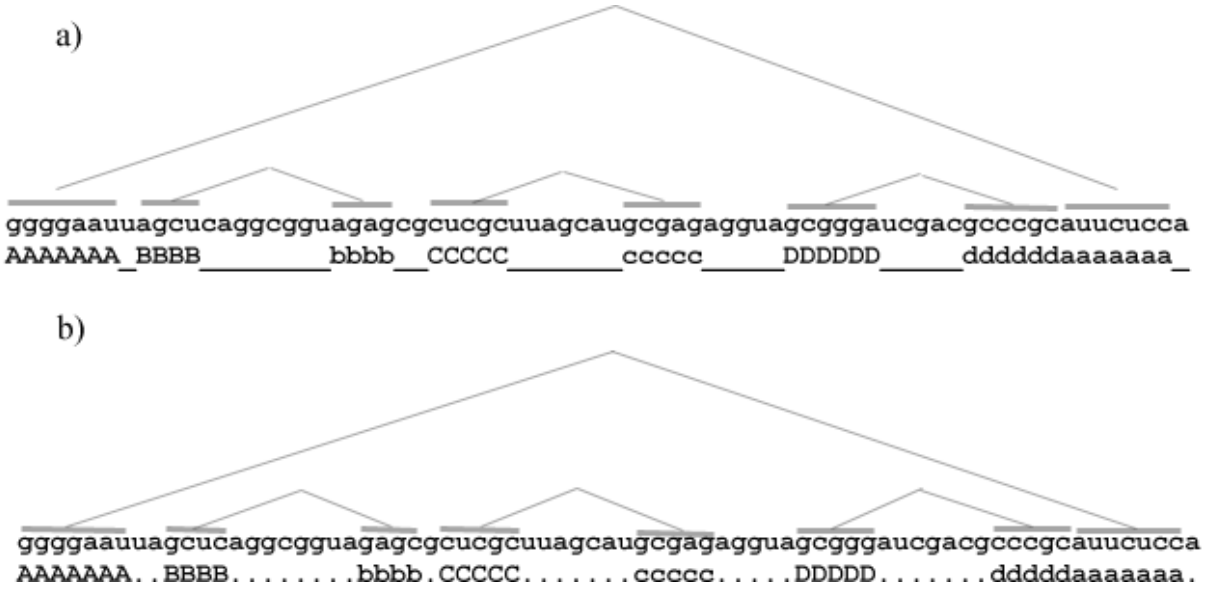


Figure 23 : a) predicted derivation tree of Ala tRNA
b) original derivation tree for the same molecule

5.3. Termination Condition

The running time of our system depends highly on the termination condition which is based on the number of generations that GA evolves. Determining the generation of convergence can be done by observing the average fitness and the fitness of the best individual for certain generations. The example below is taken from the evolution of a grammar topology with 56 rules which predicts a stem-loop structure. The population size is 100. In Figure 24, you can observe the convergence after the 60th generation when the best individual is attained and the average fitness of the population stops fluctuations.

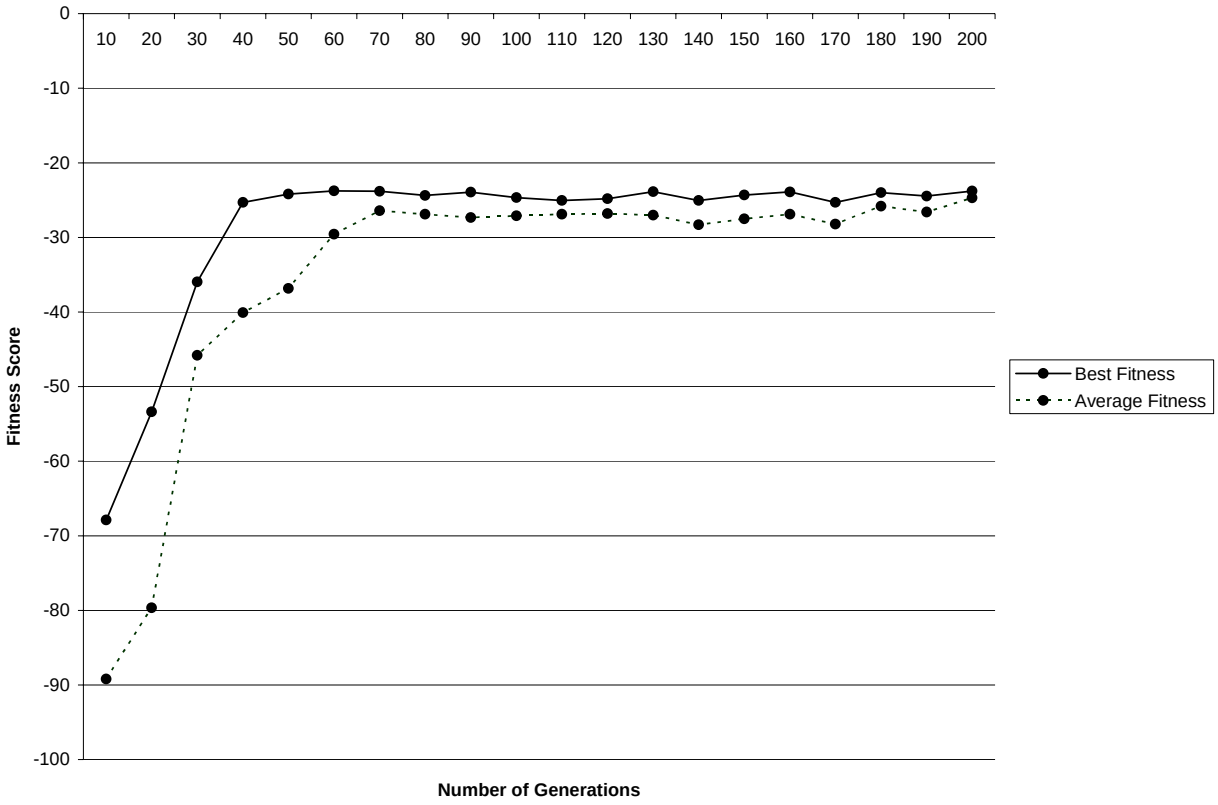


Figure 24: Fitness values converging through generations

Once convergence is observed the GA can run several times with the optimized number of generations with more training sequences simply adding their fitness values. Similarly for a grammar with similar topology, having close number of nonterminals and rules, GA can run with the optimized value to save running time for the other evolutions.

In Figure 25 the number of generations needed for convergence is shown as a function of population size. It is evident that number of generations until convergence increases by the population size both for simple and real sequences. The convergence in these experiments is determined by the generation where best individual is get and also observing minor changes after we attain the best individual. The Ala tRNA results are not shown for population sizes bigger than 350 because the running time is longer than the EM method for those high constant values.

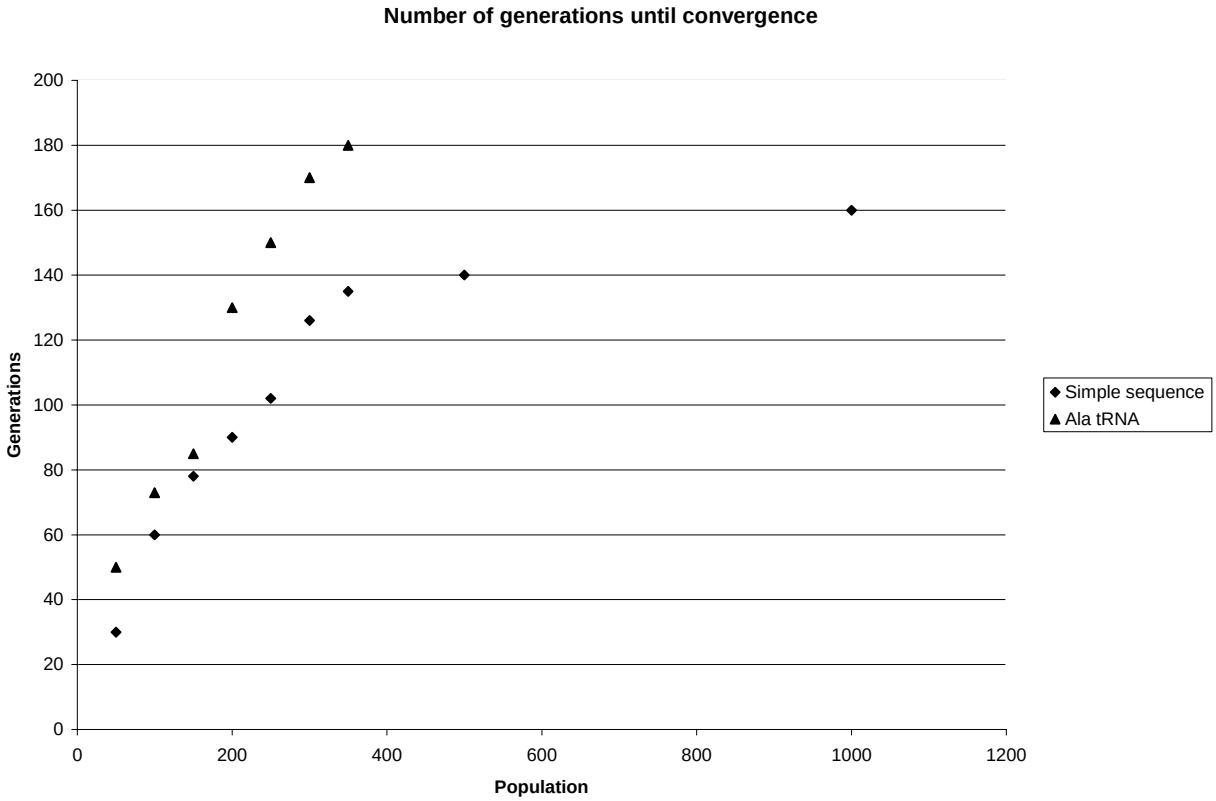


Figure 25 : Number of generations until GA converges

5.4. Grammar Construction

The construction of the context-free grammar which will be used to model the secondary structure or a consensus structure is independent from the evolution of its parameters. The grammar puts a lot of restrictions on the structure space that it models with its topology. Creating a generic grammar is not a solution to cover all possible derivations or conformations because the parameter set may converge to values that are actually average values for certain stem-loop structures. For instance if there are nested stems with lengths *m* and *n* in an RNA sequence, instead of having different rules for those stems, the generic grammar will have the parameters set with a tendency to create a single stem which has a length of *(m+n)/2* nucleotides.

More formally, if the sum of probabilities associated with the rules generating terminals with the same nonterminal on the left-hand side is p , then the rules with the same non-terminal on the left-hand side and some other nonterminals on the right-hand side will have an aggregate probability of $(1-p)$. Statistically the average length of the subsequence rooted at this nonterminal will be $1/(1-p)$.

Constructing a grammar can be approached differently according to the problem domain. If we know the structure of the group of sequences and we are trying to find their consensus structure, there are systematic ways to generate the grammar starting form the start symbol and then for each region (stem or loop) creating new nonterminals and combining them at the right-hand side of the rules. Therefore once we know the structure we can generate the grammar following the regions. Note that there are infinitely many grammars that can model the same group of sequences. Being more specific or more general depends on the amount of flexibility you want to have when you are searching for the consensus structure.

In the second problem domain where we do not know the consensus structure and we are trying to predict the secondary structure, the grammar would be constructed with a more general topology. In such a case, one GA would not be sufficient enough to be sure that the prediction is correct. Therefore out of several runs, the most similar predictions should be considered as the output since GA would have converged to a parameter set where the conformation has an optima but the prediction is not the most accurate one. This case can occur especially the population size and the number of generations are not set carefully. The similarity between the predictions can be determined by a scoring schema and the voting method can decide which prediction and which parameter set is the optimum one.

5.5. Time Issues

5.5.1. Running Time

The time complexity of our system is governed by the fitness function to find the total probability of derivation of the given sequence by Inside Algorithm. When compared to the efficient EM estimation algorithm ($O(L^3M^3)$), the complexity of our system ($O(L^3M^2)$), has a difference by a factor of number of nonterminals. Here L is the length of the sequence and M is the number of nonterminals. Note that for genetic algorithms it is nearly impossible to estimate the time complexity precisely since the constants we assume are closely related to the problem and they are subject to change. The time complexity however is not enough to make the running time comparison since the constant values differ for both algorithms. In our system, the constant values which are the population size and the number of generations can be changed in order to attain an optimum running time. When we compare the simple sequence prediction where both algorithms have 100% accuracy, our system has outperformed the EM program by the ratio of approximately 2/3. A complex structure analysis can not be compared because the space complexity of the EM algorithm was too high to run it on our test machine. On the other hand, note that the EM program using was originally designed for considering pseudoknot cases too and the grammar handling module is more complex than ours.

As we have mentioned in section 5.3 selecting the termination condition affects the running time. In addition population size should be selected to cover the search space as much as possible but we should avoid a need for a larger number of generations. The population size is selected depending on the size of the grammar and since there is no definite rule for selecting it we have done experiments comparing the accuracy percentage in the prediction in order to decide the correct size (Figure 26). The number of generations in these experiments is adjusted to observe convergence in GA.

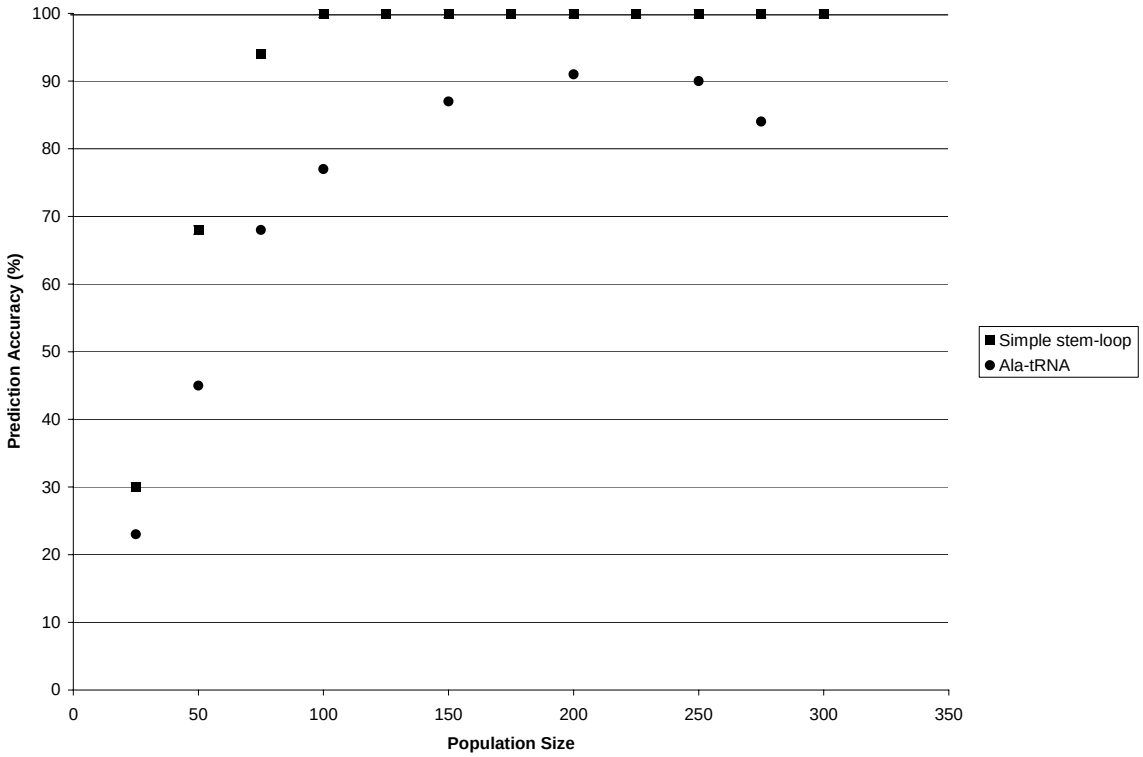


Figure 26 : Accuracy of prediction as a function of population size.

5.5.2. Optimizing the Inside Algorithm

We have investigated possible speed-up methods for the Inside Algorithm. The idea can be applied to CYK and other dynamic programming approaches that uses SCFG too. The motivation comes from the fact that the three-dimensional matrix where we store sub-solutions has some cells that do not need to be computed. This invalidity comes from the fact that the grammar is under constraint to generate certain subsequences. Therefore we can apply a preprocessing phase in order to filter out the invalid cells in the matrix.

Suppose $S_0 \to aS_1b$ is the rule with the start symbol on the left-hand side. When we fill the matrix we are calculating every subsequence that can be rooted at S_0 although we know the only subsequence that S_0 can generate is the whole sequence. Similarly we can determine the

ranges for each non-terminal by looking at the rules and eliminate the invalid cells to save computation time.

Moreover the grammar structure and the probability distribution suggest us to take advantage of heuristics about the average length of a loop. We have mentioned this property in Section 5.4. Assume there is a rule with associated probability *p,* generating a recursive loop such as $S \to aS \ \{p\}$. Since the other rules that have the same nonterminal on the left-hand side has the probability *q=1-p*, the loop generation probability is derived as:

$$q + qp^1 + qp^2 + .. + qp^n = q(1 + p^1 + p^2 + ... + p^n) = q\frac{1-p^{n+1}}{1-p} = 1 - p^{n+1}$$

By using the heuristics to bound this value to a threshold we can estimate *n,* the length of the loop from this formula. Therefore the valid cells can only be calculated for some range around the average length of the loop.

The tests done with the speed-up method resulted in ½ running time for the CYK algorithm which includes computing only the ¼ of the all possible cells in the matrix. With the speed-up of the Inside Algorithm our system will benefit with the same ratio of time gain too.

6. CONCLUSION

6.1. Concluding Remarks

In this thesis, the use of genetic algorithms in stochastic context free grammar parameter estimation is presented. The estimation is used in the domain of bioinformatics specifically RNA secondary structure prediction. Our results show that using a randomized search technique such as genetic algorithms is more flexible, scalable and efficient in terms of time and memory compared to the other best known technique with EM algorithm. On the other hand there is a trade off for accuracy in prediction, especially for complex structures.

The issue here is deciding the trade-off between time and accuracy. For complex structures we need to design complex grammars which results in large sized genotype in the GA. Since there are more parameters the convergence needs more generations and the population size is increased in that manner too. The flexibility can be an advantage in this case. If a near optimal solution can be enough for the researcher, the GA can be run for a shorter time and the best individual seen throughout the generations can be used to apply the prediction algorithm without reaching convergence. In fact this is the case where homologous sequence search is done. In that case time is a more significant issue than accuracy over a certain percentage. Our results show that we can fulfill those requirements to find near optimal solutions in the search space.

GAs can be used for any optimization problem such as ours which is to find the optimal secondary structure. The EM solution for the same problem has a tendency to fall to local optima since it starts with a single initial parameter set and iterate over those values. Since the GA starts the search aiming to cover the whole search space it is always more likely to find the global optimum.

Convergence is another aspect in a GA where the termination condition should be determined carefully. In our experiments for a certain size of a grammar, we have observed the number of generations where GA converges and then reduced the number of generations around that amount so that for other training data we can have an optimum running time for estimation.

The flexibility of this framework can be extended for prediction of RNA secondary structures with pseudoknots, protein folding modeling with grammatical approaches, homology and database search for biological sequences. Further directions are discussed in the next section. In addition, the domain can be shifted to other grammatical modeling problems other than bio-informatics where only the fitness function would need to be changed in the entire framework.

6.2. Future Directions

The GA for stochastic grammar estimation can be improved in various ways in order to accomplish faster estimation and more accurate prediction. In each paragraph we will discuss a possible EC approach for the same problem domain.

Firstly, the termination condition can be satisfied dynamically where the GA can decide when to stop the generation process. Since GAs explore the search space first and then exploit the optima regions, the convergence can be detected by observing the best individual in the population for every generation. The termination can be determined by taking the average of the best individuals (or the average fitness of the population) for a certain window size among the generations. If there is no significant change in the fitness values for those individuals, that concludes that GA is exploiting the optimum solution and there is no need iterate more. Again the accuracy-time trade-off decision will be left to the user namely the molecular biology researcher who would know the domain of the application.

Secondly, the grammar topology can be designed in a more systematic way so that it can cover all the instances in the training set and be as specific as possible. By this way, the solution to the problem of finding the class of a given sequence would not result with false positives. Since the conserved structure percentage is higher than the non-conserved parts in a group of sequences giving a false negative result has a lower probability in this case. Therefore grammar construction should be as specific as possible however it should contain every possible secondary structure and would be in optimized size in order to keep the genotype as short as possible.

Secondary structure prediction is not the only problem which can be solved with the grammatical approaches. Homology finding and database search are also important issues in the field. Suppose we have a family of related RNA's , e.g. transfer RNAs or group I catalytic introns, which share a common consensus secondary structure as well as some primary sequence, and we want to search a sequence database for homologous RNAs. The SCFG based covariance models (CM) which are counterparts of profile HMM's in HMM modeling, specify a repetitive tree-like SCFG architecture suited for modeling consensus RNA secondary structures (Eddy & Durbin, 1994; Sakakibara et. al., 1994). Using CM is a powerful tool to find the consensus structures given a family of sequences. In addition the same model can be used for classification purposes where we are trying to find which family a sequence belongs to. The grammar parameter estimation is a crucial process in this model too. Our framework is also suitable for CM because other than converging to the global optima, after a certain number of generations the parameter set will be conforming the training set that can be used to model the generic grammar. This grammar will represent the consensus structure. In addition the fitness value will indicate the amount of homology (i.e. in percentage) of the given sequence with the consensus represented by the grammar.

Stochastic context-free grammars are capable of representing most of the secondary structures for RNA sequences however they are incapable of representing pseudoknots. Cai et. al. (2003) have presented the parallel communicating grammatical systems to include structures with pseudoknots with a basic extension to SCFG modeling. Our fitness function can be modified to calculate the maximum likelihood with this approach so that our prediction can include pseudoknots too.

Fast messy GAs can be used for the same problem domain. Our work uses the heuristic crossover where the building block hypothesis is not working like the way it is applied in conventional recombination operators. Since fast messy GAs use the locus and allele information they can take the advantage of grouping rules with the same non-terminal on the left hand side. Moreover, the rules which have common non-terminals on the right hand side can be grouped as a subgroup too. The fmGA can generate the building blocks where some rule groups would result with above-average fitness values and they can be preserved.

Another future research can be using genetic programming (Koza, 1992) where instead of solving a problem, and instead of building an evolution program to solve the problem, the

evolution will search the space of possible programs for the best grammar. In genetic programming the hierarchically structured computer programs are evolved from the hyperspace of valid programs. These programs can be viewed as a space of rooted trees. From a grammatical way the trees can represent the derivation trees of biological sequences. In this case we will be creating the most fit derivation tree for a specific sequence. This tree will be consisting of rules used to form that sequence and the frequency of each rule can determine its probability parameter for the output estimation.

7. REFERENCES

Baker, J.E. (1985). Adaptive Selection Methods for Genetic Algorithms. In J. J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pages 100-111, Hillsdale, New Jersey.

Baum, L. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, 3:1-8.

Cai, L., Malmberg, R.L. and Wu, Y. (2003). Stochastic modeling of RNA pseudoknotted structures: a grammatical approach. *Proceedings of ISMB'03 and Bioinformatics* 19(s1) i66-i73.

Clark, David E. and Westhead, David R. (1996). Evolutionary algorithms in computer-aided molecular design. *Journal of Computer-Aided Molecular Design*, 10(4):337—358.

Cuff, J.A. & Barton, G.J., (1999). Evaluation and improvement of multiple sequence methods for protein secondary structure prediction. *Proteins*, v.4, pp. 508-519.

Desjarlais, J. R. and Handel T. M. (1995) . De Novo Design of the Hydrophobic Cores of Proteins . *Protein Science* **4**, 2006-2018.

Durbin, R., Eddy, S., Krogh, A, & Mitchison, G. (1998) . *Biological Sequence Analysis*, Cambridge University Press, Cambridge.

Eddy, S.R. and Durbin, R.(1994). Introduces the use of stochastic context free grammar methods for RNA sequence/structure analysis .*Nucl Acids Res.* 22:2079-2088.

Eddy, S.R. & Rivas, E. (1999). A Dynamic Programming Algorithm for RNA Structure Prediction including Pseudoknots, *J. Mol. Biol.*, 285:2053-2068.

Fogel, D. B. (2000). *Evolutionary Computation, Toward a New Philosophy of Machine Intelligence, 2 ed* .IEEE Press.

Fogel, D. B., Corne D.W. (2003). *Evolutionary Computation in Bioinformatics*. Morgan Kaufmann.

Ghirlanda, G., Lear, J.D., Lombardi, A. and DeGrado, W.F. (1998). From synthetic coiled coils to functional proteins: automated design of a receptor for the Calmodulin-binding domain of calcineurin. *J. Mol. Biol.*, 281, 379-391.

Goldberg, D.E. (1990). A note on Boltzmann tournament selection for genetic algorithms and populationoriented simulated annealing, *Complex Sys.* 4, 445-460.

Goldberg, D. E. (1989). *Genetic Algorithms in Search*, *Optimization, and Machine Learning*. Addison-Wesley, Reading, MA,

Goldberg, D.E., Deb, K., Kargupta, H. and Harik, G. (1993). Rapid accurate optimization of difficult problems using fast messy genetic algorithms. In S. Forrest, editor, *Proc. 5th Int'l Conference on Genetic Algorithms*, pages 56-64.

Greenwood, G. (1999). Revisiting the Complexity of Finding Globally Minimum Energy Configurations in Atomic Clusters. *Zeitschrift für Physikalische Chemie* Vol. 211, 105-114.

Guigo R., Knudsen S., Drake N., and Smith T. (1992). Prediction of gene structure. *J. Mol. Biol.* 226, 141-157**.**

Holland,J. (1975). *Adaption in natural and artificial systems*, The University of Michigan Press.

Holland, J. H. (1992). Adaptation in natural and artificial systems: an introductory analysis with applications in biology, control and artificial intelligence, *Complex adaptive systems*, MIT Press, Cambridge, MA.

Konig R, Dandekar T. (1999). Improving genetic algorithms for protein folding simulations by systematic crossover. *Biosystems.*, 50(1):17-25.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge, MA, USA: MIT Press.

Landavazo, D.G., Fogel, G.B., and Fogel, D.B.. (2002). Quantitative Structure-Activity Relationships by Evolved Neural Networks for the Inhibition of Dihydrofolate Reductase by Pyrimidines. *BioSystems*, Vol. 65:1, pp. 37-47.

Lari, K. and Young, S. J. (1990). The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 4:35-56.

Lowe, T. and Eddy, S.R. (1997). tRNAscan-SE: a Program For Improved Detection of Transfer RNA genes in Genomic Sequence *Nucl. Acids Res.*, 25:955-964.

Michaud, S. R., Zydallis, J. B., Strong, D.M. and Lamont, G. B. (2001) Load Balancing Search Algorithms on a Heterogeneous Cluster of PCs. IN *Tenth SIAM conference on Parallel Processing for Scientific Computing-pp01,* Portsmouth, Virginia.

Morris, G.M., Goodsell, D.S., Halliday, R.S., Huey, R., Hart, W.E., Belew, R.K. and Olson, A.J. (1998). Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function. *J. Comput. Chem.*, 19(14):1639--1662, (Goldberg et al., 1989)

N. Chomsky. (1959). On certain formal properties of grammars. *Information and Control,* 2(2):137—167.

Notredame,C. Holm,L. and Higgins,D.G. (1998) . COFFEE: an objective function for multiple sequence alignments. *Bioinformatics,* 5, 407-422.

Porto, V. W., Fogel, D. B. and Fogel, L. J. (1995). Alternative Neural Network Training Methods, *IEEE Expert*, volume 10, no.4, pp. 16-22.

Qian N, Sejnowski T J. (1988). Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology*, 202, 865-884.

Qian, N. and Sejnowski, T. J. (1988). Predicting the secondary structure of globular proteins using neural network models. *J. Mol. Biol.*, 202:865-884.

Riis S K, Krogh A. (1996). Improved prediction of protein secondary structure using structured neural networks and multiple sequence alignments. *Journal of Computational Biology*, 3, 163-183.

Ripley B D (1996): *Pattern Recognition and Neural Networks*, Cambridge University Press.

Rost B, Sander C (1993). Prediction of protein secondary structure at better than 70% accuracy. *Journal of Molecular Biology*, **232**, 584-599.

Sakakibara, Y. (1992). "*Efficient Learning of Context-Free Grammars from Positive Structural Examples*", Information and Computation 97, p23-60.

Sakakibara, Y., Brown, M., Hughey, R., Mian, I. S., Sjolander, K., Underwood, R. C. and Haussler, D. (1994) . Stochastic Context-Free Grammars for tRNA Modeling, *Nucleic Acid Res.* 22, 5112-5120.

Sakakibara, Y., Kondo, M. (1999). Ga-based Learning of Context-free Grammars Using Tabular Representations. *Proceedings of 16th International Conference on Machine Learning (ICML-99)*. 354-360.

Schaffer, D. (1987). Some effects of selection procedures on hyperplane sampling by genetic algorithms. *Genetic Algorithms and Simulated Annealing*, chapter 7, pages 89--103. Morgan Kauffmann Publishers, Inc., Los Altos, California.

Searls, D.B. (1993). The Computational Linguistics of Biological Sequences. In L. Hunter, editor, *Artificial Intelligence and Molecular Biology*, pages 47-120. AAAI Press.

Shapiro, B.A. and Navetta, J. (1994). A massively parallel genetic algorithm for RNA secondary structure prediction. *The Journal of Supercomputing*, Volume 8, pp. 195-207.

Shapiro, B.A. and Wu, J-C. (1997). Predicting RNA H-type pseudoknots with the massively parallel genetic algorithm. *CABIOS*, 13(4): 459-471.

Shapiro, B.A. and Wu, J-C.(1996). An annealing mutation operator in the genetic algorithms for RNA folding. *CABIOS*, 12(3): 171-180.

Shapiro, B.A. and Wu, J-C., Bengali, D., and Potts, M. (2001). The massively parallel genetic algorithm for RNA folding: MIMD implementation and population variation. *Bioinformatics* 17(2): 137-148.

Snyder,E.E. and Stormo,G.D. (1995) . *Identification of protein coding regions in genomic DNA*. J. Mol. Biol. 248, 1--18.

Socci,N., Bialek, W., and Onuchic, J.N. (1994). Properties and origins of protein secondary structure. *Phys Rev E* 49, 3400-3443.

Sun, Z., Xia, X., Guo, Q. and Xu, D. (1999). Protein Structure Prediction in a 210-type Orthogonal Lattice Model: Parameter Optimization in the Genetic Algorithm using Orthogonal Array. *J. Protein Chemistry* , 18, 39-46.

Uberbacher, E. C., Xu, Y. and Mural R. J. (1995). Discovering and Understanding Genes in Human DNA Sequence Using GRAIL. *Computer Methods for Macromolecular Sequence Analysis*. September 1995.

Whitley, D. (1989). The genitor algorithm and selection pressure. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116--121. Phillips Laboratories, Morgan Kaufmann Publishers, Inc.

Wilson, C., Gregoret, L. and Agard, D. (1993). Modeling side-chain conformation for homologous proteins using an energy-based rotamer search. *J. Mol. Biol.*, 229:996-1006.

Wright, A. (1991). *Genetic Algorithms for Real Parameter Optimization*. Foundations of Genetic Algorithms 1, G.J.E. Rawlin (Ed.), (Morgan Kaufmann, San Mateo), p.205 - 218.

Yao X. (1999). Evolving Artificial Neural Networks. *Proceedings of the IEEE, 87*, pp. 1423-1447.

Zuker, M. (1989). On Finding All Suboptimal Foldings of an RNA Molecule*, Science*, 244:48-52.

APPENDIX A

Source Code – GA Driverx

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <time.h>
#include <ga/ga.h>      // we're going to use the steady state GA
#include <ga/GARealGenome.h>
#include <ga/GARealGenome.C>

using namespace std;

#define POPSIZE 50 //population size in GA
#define NGEN 50 //number of generations in the evolution
#define PMUT 0.005 //mutation probability
#define PCROSS 0.9 //crossover probability
#define NREP 0.6 //Replacement percentage
#define MAXRULES 50 //Maximum number of rules in the given grammar
#define MAXNONTERMINALS 30 //maximum number of nonterminals in the given
grammar
#define MAXRNALENGTH 200 //maximum lenght of the RNA sequence
#define MINIMUM -10000 //minimum value to initialize the matrix
#define MAXLINE 80 //max characters in a line to read input
#define MAXBASENAME 20 //maximum characters for a base name
#define MAXHEURISTICTRY 30 //maximum trials for heuristic crossover

float Objective(GAGenome &); // This is the declaration of our obj function.
//structure holding chomsky first type rules
struct strule1 {

      int lhs;
      char rhs;
      float prob;
      int rulenum;
};
//structure holding chomsky second type rules
struct strule2 {

      int lhs;
      int rhs1;
      int rhs2;
      float prob;
      int rulenum;
};
//structure for CYK traceback
struct sttrack {
      int rule;
      int k;
};

struct stfindprob {
```

```
	int rule;
	float prob;
};


char names[MAXRULES][MAXBASENAME]; //first part are nonterminals second part
are terminals
strule1 rules1[MAXRULES]; //Chomsky in form X-> t
strule2 rules2[MAXRULES]; //Chomsky in form X-> YZ

/* DP DATA STRUCTURES*/

float M[MAXNONTERMINALS][MAXRNALENGTH][MAXRNALENGTH];
sttrack S[MAXNONTERMINALS][MAXRNALENGTH][MAXRNALENGTH];

char rna[MAXRNALENGTH]="";
char rnapaired[MAXRNALENGTH]="";
int rnalength=0;
int numsecondtype=0;
int numfirsttype=0;
int numrules=0;
int numnonterminals=0;
char rnafilename[MAXLINE]="seq.txt";
char gramfilename[MAXLINE]="scfg.txt";
char *outfilename= "out.txt";
FILE *outfile;
char *rnaoutfilename= "ann.txt";
FILE *rnaoutfile;


int nonterminalindex[MAXNONTERMINALS]; //used in GA Reparing to hold the
index value for each nonterminal-rhs should sum to 1

int findnonterminal(char *token) //takes nonterminal string returns its
number
{
	int i=0;
	while (strcmp(names[i],token))
		i++;
	return i;

}

int findterminal(char a) //takes terminal character returns its number if it
can not find adds it
{
	int i=numnonterminals;

	if (a>='A' && a<='Z')
			a =  a + ('a'-'A');

	while (names[i][0]>='a' && names[i][0]<='z')
	{
		if (names[i][0]==a)
			return i;
		i++;
	}
```

```
      names[i][0]=a;
      names[i][1]='\0';

      return i;

}

void readfiles() //reads the sequence and grammar files
{

      FILE *rnafile, *gramfile;
      char readrna[MAXRNALENGTH];
      char *token;
      char line[MAXLINE];
      char seps[] = " ;{},\t\n";

      int index=0;
      int lhs;
      int rhs1;
      int rhs2;
      float prob;
      int cnrule1=0;
      int cnrule2=0;
      int cnrules=0;
      //int cnnonterminals=0;

   if( (rnafile  = fopen( rnafilename, "r" )) == NULL )
      printf( "The file 'rna' was not opened\n" );
   else
      printf( "The file 'rna' has opened\n" );

      fscanf( rnafile, "%s", readrna );


      rnalength= strlen(readrna);


      for (int i=0; i<rnalength;i++)
      {
            if (readrna[i]<='Z')
                  readrna[i] =  readrna[i] + ('a'-'A');
      }

      strcpy(rna,readrna);



   if( fclose( rnafile ) )
      printf( "The file 'rna' was not closed\n" );
    //****************GRAMMAR FILE******

   if( (gramfile  = fopen( gramfilename, "r" )) == NULL )
      printf( "The file 'grammar' was not opened\n" );
   else
      printf( "The file 'grammar' has opened\n" );
```

```
while (fgets( line,MAXLINE, gramfile ))
{

      token = strtok( line, seps );


      if (token[0]=='<')
      {

            strcpy(names[numnonterminals],token);
            nonterminalindex[numnonterminals]=index;
            numnonterminals++;
      }
      index++;

}
nonterminalindex[numnonterminals]=index;
rewind(gramfile);

while (fgets( line,MAXLINE, gramfile ))
{
      cnrules++;

      token = strtok( line, seps );
      if (token[0]=='<')
      {
            lhs= findnonterminal(token);
            token = strtok( NULL, seps ); //read '|'
      }

      while( token != NULL )
      {
             token = strtok( NULL, seps );

            if (token[0]>='A' && token[0]<='a')
            {
                  rhs1= findterminal(token[0]);
                  token = strtok( NULL, seps );
                  prob= atof(token);
                  rules1[cnrule1].lhs=lhs;
                  rules1[cnrule1].rhs=rhs1;
                  rules1[cnrule1].prob=prob;
                  rules1[cnrule1].rulenum=cnrules-1;
                  cnrule1++;
            }
            else

            {
                  rhs1= findnonterminal(token);
                  token = strtok( NULL, seps );
                  rhs2= findnonterminal(token);
                  rules2[cnrule2].lhs=lhs;
                  rules2[cnrule2].rhs1=rhs1;
                  rules2[cnrule2].rhs2=rhs2;
                  rules2[cnrule2].rulenum=cnrules-1;
```

```
                        token = strtok( NULL, seps );
                        prob= atof(token);
                        rules2[cnrule2].prob=prob;

                        cnrule2++;

                  }

                  token = strtok( NULL, seps );
            }
      }

      numrules=cnrules;
      numfirsttype=cnrule1;
      numsecondtype=cnrule2;

      if( fclose( gramfile ) )
            printf( "The file 'grammar' was not closed\n" );


}

stfindprob findprob(int i,int j,int l,int k)
//The total probability of ith nonterminal to create rna[j..k] and
rna[k+1..l] for every rule which has i on lhs.

{
      int a=0;
      int X=i;
      int Y=0;
      int Z=0;
      stfindprob result;
      float curprob=MINIMUM;
      int maxrule=0;
      float maxprob=0;

      for (a=0; a<numsecondtype; a++)
            if (rules2[a].lhs==i)
            {
                  Y=rules2[a].rhs1;
                  Z=rules2[a].rhs2;

                  curprob= M[Y][j][k]*M[Z][k+1][l]*rules2[a].prob;
                  maxprob += curprob;



            }
      result.prob=maxprob;
      result.rule=maxrule;
      return result;

}

int findbaserule(int nonterminal,char a) //returns the number of rule which
is nonterminal-->a
{
```

```
      int i=0;
      for (i=0;i<numfirsttype;i++)
            if ((names[rules1[i].rhs][0] == a)&&
(rules1[i].lhs==nonterminal))
            {
                  return i;

            }

      return 0;



}

float findbaseprob(int nonterminal,char a) //returns the probability of a
rule nonterminal-->a
{

      int i=0;
      float prob=0.0;
      for (i=0;i<numfirsttype;i++)
            if ((names[rules1[i].rhs][0] == a)&&
(rules1[i].lhs==nonterminal))
            {
                        prob = rules1[i].prob;
                        return prob;



            }

      return MINIMUM; //not reached normally


}

void initialize()
{
      int i=0;
      int j=0;
      int l=0;
      //X->A diagonal is computed for base rules the other values are minimum
value
      for (i=0; i<numnonterminals;i++)
            for (j=0; j<rnalength; j++)
                  for (l=j; l<rnalength; l++)
                        if (l==j)
                        {
                              M[i][j][l]= findbaseprob(i,rna[j]);
                              S[i][j][l].rule= findbaserule(i,rna[j]);
                        }
                        else M[i][j][l] = 0;


}
```

```
void printRNA(int i, int j) //prints RNA sequence from i to j
{
      fprintf(outfile,"RNA[%d..%d]",i,j);
      for (i;i<=j;i++)
            fprintf(outfile,"%c",rna[i]);

}

//this is the Inside algorithm

float
Objective(GAGenome& g) {
  GARealGenome& genome = (GARealGenome&)g;

      int i=0;
      int j=0;
      int k=0;
      int l=0;
      int m=0;

      float maxprob=0;
      stfindprob curstruct;
      float sum=0;
      float score=0;
      float factor=0;
      float temp;

      //REPAIRING INDIVIDUAL************************************


      for (i=0; i<numnonterminals; i++)
      {
         for (j=nonterminalindex[i]; j<nonterminalindex[i+1]; j++)
         {
               sum+=genome.gene(j);

         }

         factor = (float)1/sum;

         for (j=nonterminalindex[i]; j<nonterminalindex[i+1]; j++)
         {
          temp= genome.gene(j);
            genome.gene(j, temp*factor) ;
         }


         sum=0;

      }

      //COPYING INDIVIDUAL TO THE CURRENT GRAMMAR STRUCTURE
      for (i=0; i<numrules; i++)
      {
            for (j=0;j<numfirsttype;j++)
                  if (rules1[j].rulenum==i)
```

```
                        rules1[j].prob= genome.gene(i);
          for (j=0;j<numsecondtype;j++)
                        if (rules2[j].rulenum==i)
                              rules2[j].prob= genome.gene(i);


      }


      initialize();

      //MATRIX COMPUTATION

      for (m=1;m<rnalength;m++) //m is the difference between row and column
      {
            for (i=0;i<numnonterminals;i++)  //nonterminal slice
                  for (j=0;j+m<rnalength;j++) //row
                  {
                        //Update for Inside: Comment out parts were from CYK
                        for (k=j;k<j+m;k++) //trying to find j+m'th column
trying each k inbetween
                        {

                              curstruct = findprob(i,j,j+m,k);
                              maxprob += curstruct.prob;

                        }


                        M[i][j][j+m]=maxprob;

                        maxprob=0;

                  }
      }


      //printf("Inside result for current individual:%f\n",M[0][0][rnalength-
1]);


      score = (float) M[0][0][rnalength-1];
      return score;
}


/*
Offspring1 = BestParent + r * (BestParent - WorstParent)
Offspring2 = BestParent

  It is possible that Offspring1 will not be feasible.
  This can happen if r is chosen such that one or more of
  its genes fall outside of the allowable upper or lower bounds.
  For this reason, heuristic crossover has a user settable parameter
(MAXHEURISTICTRY)
  for the number of times to try and find an r that results in a feasible
chromosome.
```

```
  If a feasible chromosome is not produced after n tries, the WorstParent is
returned
  as Offspring1.

*/
int
HeuristicCrossover(const GAGenome& g1, const GAGenome& g2, GAGenome*
c1,GAGenome* c2)
{
  GARealGenome& mom = (GARealGenome&)g1;
  GARealGenome& dad = (GARealGenome&)g2;
  GARealGenome& sis = (GARealGenome&)*c1;
  GARealGenome& bro = (GARealGenome&)*c2;
  GARealGenome& best = mom;
  GARealGenome& worst = dad;
  int counter=0;
  int feasable=0;
  float r = GARandomFloat();
  float newvalue;


  if (mom.evaluate()<dad.evaluate())
  {
         best = mom;
         worst = dad;
  }
  else
  {

        best = dad;
        worst = mom;
  }

  sis= best;


  while (counter<MAXHEURISTICTRY)
  {
         feasable=1;
         for (int i=0; i<numrules; i++)
         {
                  newvalue= best.gene(i) + r * (best.gene(i) -
worst.gene(i));
                if (newvalue>=0 && newvalue<=1)
                        bro.gene(i, newvalue);
                  else {
                        feasable=0;
                        break;
                  }
         }

         if (feasable)
               return 1;
         r =      GARandomFloat();

         counter++;
  }
```

```
  if (!feasable)
        bro = worst;
  printf("No feasable offspring from heuristic crossover\n");

  return 1;
}



void main(int argc, char **argv)
{
      int i=0;
      int j=0;
      int k=0;
      int l=0;
      int m=0;
      char a;
      float maxprob=MINIMUM;

      int maxk=MINIMUM;


      float sumprob=0;

      int popsize  = POPSIZE;
      int ngen     = NGEN;
      float nrep = NREP;
      float pmut   = PMUT;
      float pcross = PCROSS;
      clock_t start, finish;

//    cout<<"Enter RNA file name:"<<endl;
//    cin>>rnafilename;

//    cout<<"Enter grammar file name:"<<endl;
//    cin>>gramfilename;

      printf("The file names are hardcoded...\n");

      readfiles();
      printf("The length of the gene is %d\n",numrules);


      for(int ii=1; ii<argc; ii++) {
            if(strcmp(argv[ii++],"seed") == 0) {
                        GARandomSeed((unsigned int)atoi(argv[ii]));
            }
      }

// Declare variables for the GA parameters and set them to some default
values.
// Now create the GA and run it.  First we create a genome of the type that
// we want to use in the GA.  The ga doesn't operate on this genome in the
// optimization - it just uses it to clone a population of genomes.
```

```
   GARealAlleleSet alleles(0, 1);
   GARealGenome genome(numrules, alleles, Objective);

   genome.crossover(HeuristicCrossover);
   //genome.crossover(GARealBlendCrossover);
// Now that we have the genome, we create the genetic algorithm and set
// its parameters - number of generations, mutation probability, and
crossover
// probability.  And finally we tell it to evolve itself.

  GAParameterList params;
  GASteadyStateGA::registerDefaultParameters(params);
  params.set(gaNpopulationSize, popsize); // population size
  params.set(gaNpCrossover, pcross);      // probability of crossover
  params.set(gaNpMutation, pmut);   // probability of mutation
  params.set(gaNnGenerations, ngen);      // number of generations
  params.set(gaNpReplacement, nrep );     // how much of pop to replace each
gen
  params.set(gaNscoreFrequency, 10);      // how often to record scores
  params.set(gaNflushFrequency, 10);      // how often to dump scores to file
  params.set(gaNscoreFilename, "rnaga.dat");

  GASteadyStateGA ga(genome);
  ga.parameters(params);
  printf("\nGA has started......\n");

  start = clock();
  ga.evolve();
  finish = clock();

  double duration = (double)(finish - start) / CLOCKS_PER_SEC;



   genome = ga.statistics().bestIndividual();
   //COPYING INDIVIDUAL TO THE CURRENT GRAMMAR STRUCTURE

    for (i=0; i<numrules; i++)
      {
            for (j=0;j<numfirsttype;j++)
                  if (rules1[j].rulenum==i)
                        rules1[j].prob= genome.gene(i);
        for (j=0;j<numsecondtype;j++)
                  if (rules2[j].rulenum==i)
                        rules2[j].prob= genome.gene(i);


      }

      //CREATING OUTPUT FILE
   outfile = fopen( outfilename, "w" );

   fprintf(outfile,"rna %s\n",rna);
   fprintf(outfile,"Popsize:%d Ngenerations:%d replacement:%f mutation
prob:%f cross.prob:%f\n",popsize,ngen,nrep,pmut,pcross);
```

```
   fprintf(outfile,"FIRST TYPE \n");
   for (i=0;i<numfirsttype;i++)
      {
            k=rules1[i].lhs;
            l=rules1[i].rhs;
            fprintf(outfile,"%s\t|%s
{%f}\n",names[k],names[l],rules1[i].prob);
      }


      fprintf(outfile,"SECOND TYPE \n");
      for (i=0;i<numsecondtype;i++)
      {
            k=rules2[i].lhs;
            l=rules2[i].rhs1;
            m=rules2[i].rhs2;
            fprintf(outfile,"%s\t|%s %s
{%f}\n",names[k],names[l],names[m],rules2[i].prob);
      }


   printf("\nBest Individual is: \n");
   for (i=0; i<numrules; i++)
   {
            printf("%d)%f\n",i+1,genome.gene(i));

   }


   maxprob = genome.evaluate();
   printf("Max prob in log is: %f\n\n",maxprob);
   fprintf(outfile,"\nInside prob in log is: %f\n\n",maxprob);
   fprintf(outfile, "It took %2.3f seconds to run the GA\n", duration );

   fclose(outfile);

      printf("GA FINISHED...\n");
      cin>>a;


}
```