

# INTELLIGENT RNA FOLDING WITH ATOMIC FORCE MICROSCOPY INFORMATION

by

**Asim Khan**

A thesis submitted in partial fulfilment of the requirements  
for the degree of

**MSc Computer Science**



**Lincoln College**

**University of Oxford**



September 2005

**Supervisors:**

**Professor Jotun Hein (Oxford Centre of Gene Function)  
Professor Peter Jeavons (Computing Laboratory)**

## ABSTRACT

*The area of RNA secondary structure prediction either by free energy minimization or comparative analysis has been extensively researched for over two decades. Several techniques have been developed constantly showing a nuanced improvement in the notion of accuracy in the field. Here we describe an enhancement to the process by utilising the technology of microbial imaging.*

*The advancement in the Atomic Force Microscopy imaging technology has led us today to sub-nanometre resolution images of RNA molecules. The detailed information that we can possibly acquire from these images can enrich us with valuable constraints that can play a vital role in considerably shrinking the enormous search space involved in RNA folding mechanism.*

*A research was conducted to analyse the present RNA structure prediction techniques in order to select an appropriate algorithm which is capable of incorporating this additional information. A genetic algorithm was selected for implementation and its fitness evaluation function was enhanced to accept variable constraint based input. A more traditional Nussinov algorithm was also utilised in the initial phases of the research mainly as a learning tool to grasp the basic mechanism of RNA structure prediction process.*

*Several sample test cases were selected and in some cases even generated specifically to test the effectiveness of the application of these constraints to the folding mechanism. At an early phase of this research, the results hint out that these additional constraints simulating the real life images of the molecules themselves, do make a difference in defining the dimensions of the predicted secondary structure of a respective RNA molecule, hence opening research doors for this novel dimension to the rather exhaustive RNA structure prediction research.*

## DECLARATION

I, Asim Khan hereby declare that the content of this dissertation is entirely my own work (except where otherwise indicated), that it has not been submitted for a degree of any other university, and that all the assistance I have received has been fully acknowledged.

Signature .....

College: Lincoln

Date: 2 September 2005

## ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to Professor Hein and Professor Jeavons for invaluable guidance throughout the course of the project. Thanks to Ebbe Sloth Andersen of Aarhus University, Denmark for making some useful AFM images from his research available to assist initiating the research thought process. In addition, special thanks to the OCGF team whose familiarity with the needs and ideas of the research was helpful during the early programming phase of this undertaking. Their extremely motivating weekly Breakfast club presentations not only provided state of the art information on the current research but was always a priceless opportunity to raise and discuss project issues. Thanks also to the members of the faculty for their constructive input and last but not least special thanks to the friends and family for their moral support throughout my time at Oxford.

# TABLE OF CONTENTS

<b>INTRODUCTION TO THE PROBLEM.....</b>	<b>1</b>
THE RNA SECONDARY STRUCTURE AND ITS SIGNIFICANCE.....	1
RNA SECONDARY STRUCTURE PREDICTION .....	2
A NEW DIMENSION TO THE PROBLEM.....	3
ALGORITHMIC APPROACH.....	5
RESEARCH AIM.....	6
<b>ATOMIC FORCE MICROSCOPY IMAGING OF RNA.....</b>	<b>7</b>
INTRODUCING ATOMIC FORCE MICROSCOPY .....	7
THE IMPORTANCE OF AFM INFORMATION IN BIOINFORMATICS .....	8
AFM IN RNA STRUCTURE PREDICTION WORLD .....	8
THE GULF TO BRIDGE.....	10
<b>EXPLORING RNA FOLDING TECHNIQUES.....</b>	<b>11</b>
A MATHEMATICAL MODEL OF THE RNA SECONDARY STRUCTURE.....	11
RNA FOLDING AS A SEARCH PROBLEM.....	15
EXPLORING SEARCH ALGORITHMS .....	16
REDUCING THE SEARCH SPACE .....	17
EARLY HISTORY OF RNA STRUCTURE PREDICTION .....	17
APPLICATION OF INTELLIGENT SYSTEMS.....	17
MACHINE LEARNING TECHNIQUES .....	18
ARTIFICIAL NEURAL NETWORKS .....	18
GENETIC ALGORITHMS.....	19
SELECTING AN ALGORITHM FOR THE PROJECT .....	21
NUSSINOV ALGORITHM FOR RNA FOLDING.....	21
GENETIC ALGORITHM FOR RNA FOLDING .....	22
<b>DESIGNING AND IMPLEMENTING THE ALGORITHMS.....</b>	<b>23</b>
PROJECT LIFECYCLE .....	23
IMPLEMENTING NUSSINOV .....	24
Design details.....	25
Implementation details.....	25
IMPLEMENTING GARNAFOLD.....	25
Design Details .....	26
Implementation details.....	26
Planning a Chromosome.....	28
Choosing a Fitness Function.....	29

Creation and Evolution of the Population .....	29
USING INFORMATION FROM THE AFM IMAGES .....	30
<b>EVALUATING THE ALGORITHMS.....</b>	<b>31</b>
EVALUATION STRATEGY .....	31
SELECTING APPROPRIATE TEST CASES.....	32
CARRYING OUT THE TESTS.....	32
ANALYSIS OF THE RESULTS .....	39
ISSUES ENCOUNTERED .....	40
<b>CONCLUSIONS.....</b>	<b>41</b>
PROJECT REVIEW .....	41
CONCLUDING REMARKS.....	42
THINKING FORWARD .....	43
MSC COURSE RELEVANCE TO THIS PROJECT.....	43
<b>BIBLIOGRAPHY.....</b>	<b>44</b>
<b>APPENDIX A – NUSSINOV ALGORITHM.....</b>	<b>48</b>
<b>APPENDIX B – TEST SEQUENCES USED.....</b>	<b>50</b>
<b>APPENDIX C – NUSSINOV SOURCE CODE.....</b>	<b>52</b>
<b>APPENDIX D – GARNAFOLD SOURCE CODE .....</b>	<b>58</b>

## LIST OF FIGURES

<i>Number</i>	<i>Page</i>
Figure 1 - A sample RNA Secondary Structure (SRPDB, 2005).....	1
Figure 2 - A sample picture of HIV RNA of 744 nucleotides.....	4
Figure 3 - Zoomed image of a single HIV RNA molecule .....	4
Figure 4 - Pictorial representation of the problem.....	6
Figure 5 - Atomic Force Microscope (Google, 2005) .....	7
Figure 6 - AFM HIV-RNA image in comparison to the computer generated structures.....	9
Figure 7 - Basic RNA structures (Aebi et al., 1996).....	13
Figure 8 - Nussinov ways to extend an optimal substructure.....	21
Figure 9 - System development lifecycle .....	23
Figure 10 - Nussinov Class diagram .....	24
Figure 11- Class diagram for GARNAFold .....	26
Figure 12 - Genetic Algorithm of GARNAFold.....	27

Figure 13 - Folded Test Sequence 1 utilising the maximum stem length .....	32
Figure 14 - Test 1 constrained Structure generated by GARNAFold.....	33
Figure 15 - A flawed structure generated by Nussinov algorithm for test case 2 .....	34
Figure 16 - Test 2 result from GARNAFold and MFOLD.....	34
Figure 17 - Test 2 result after applying constraints .....	35
Figure 18 - Test 3 structure from GARNAFold .....	36
Figure 19 - Test 3 result with a constrained stem of length 4.....	36
Figure 20 - Test 4: First of the optimal structures.....	37
Figure 21 - Test 4: Second optimal structure .....	37
Figure 22 - Test 5 resulting structure .....	38
Figure 23 - Presumed AFM image of the RNA structure of Test 5.....	39

## LIST OF GRAPHS

<i>Number</i>	<i>Page</i>
GRAPH 1 - FITNESS OPTIMISATION FOR TEST CASE 1 .....	33
GRAPH 2 - FITNESS OPTIMISATION PLOT FOR TEST 2 SEQUENCE .....	35
GRAPH 3 - COMPARISON SHOWING THE EFFECT OF LENGTH ON OPTIMISATION TIME .....	40

## Chapter 1

### INTRODUCTION TO THE PROBLEM

Genes are made up of deoxyribonucleic acid (DNA). The DNA molecule carries the genetic information in all organisms other than some viruses. Ribonucleic acid (RNA) is another class of Nucleic acids. RNA molecules function primarily in protein synthesis, acting in one capacity as messengers carrying information from instructions coded into DNA to the ribosomal sites of protein synthesis in the cell (Elrod and Stansfield, 2002).

#### The RNA Secondary Structure and its Significance

As a linear nucleic acid polymer, RNA structure has a lot of similarity to DNA structure. It is a polymer of four different nucleotide subunits, Adenine (A), Cytosine (C), Guanine (G) and Uracil (U) which is a replacement for Thymine (T) in a DNA. This sequence of bases is known as the *primary structure* of RNA, which distinguishes one RNA molecule from another. A-U and G-C form hydrogen bonded base pairs and are said to be complementary. A G-C pair forms three hydrogen bonds and is more stable than an A-U pair which forms only two such bonds. Base pairs exist on the same plane and are always stacked onto other base pairs in a RNA structure. Contiguous stacked base pairs are called stems. These short single stranded RNA molecules are much freer to fold into a coiled and looped structure called the *secondary structure* of RNA and the conformation of the molecule in 3-dimensional space is called the *tertiary structure* (Elrod and Stansfield, 2002; Durbin et al., 1998).

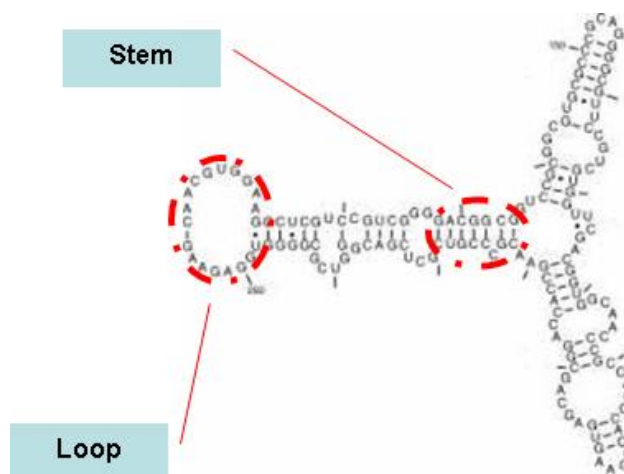


Figure 1 - A sample RNA Secondary Structure (SRPDB, 2005)

*Figure 1* shows a sample RNA secondary structure showing the base pairs, the stems and the loops. The primary structure of the RNA molecule contains all of the information for self-assembly into complex 3-D structures.

There are two biological purposes of RNA. A DNA molecule contains the genetic information, the set of instructions directing the maintenance of the cell, its growth, its differentiation, and proliferation. RNA acts as a messenger transmitting this information from DNA into the production of proteins, the catalysts and building blocks of cells and secondly it also acts as a structural component of ribosomes and other complexes. It is the secondary structure, and the resulting tertiary structure, that determine how the RNA will interact and react with other cell components (Durbin et al., 1998).

### **RNA Secondary Structure Prediction**

The problem of RNA structure prediction also known as RNA folding is one of the most critical in the field of molecular biology. Although it is possible to determine it directly by x-ray diffraction, the process is difficult, slow and very expensive. In RNA the secondary structure elements are notably more stable and develop faster than the tertiary interactions. Therefore, a separation of an RNA folding model into secondary (properly nested base pairs), tertiary (non-planar nucleotide contacts), and 3D (atom coordinate) phases seems feasible (Evers and Giegerich, 2001). Determining the secondary structure of an RNA molecule is widely seen as a first step towards understanding its biological function (Tinoco and Bustamante, 1999). Therefore, mathematical models of prediction have been developed over the years and these have led to serial as well as parallel algorithms. Single sequence RNA secondary structure prediction algorithms have already been successfully utilised since Zuker and Stiegler (1981) from the early eighties.

There are several methods to predict the RNA structure. Knowing the primary structure, the goal is to identify its function. The first step in achieving this is to predict its secondary structure. The secondary structure of RNA can expose biologically relevant features. RNA secondary structures are important in many biological processes and efficient RNA structure prediction is essential in many experiments. There are two main approaches dominant in the area; energy minimisation and comparative methods (Durbin et al., 1998).

Most of the RNA prediction methods are based on free energy estimates. The fold with more negative free energy is more stable, since it releases more stored energy. The free

energy of a fold is the addition of free energy of all the motifs found in the structure. This works with single sequences. One of the most sophisticated secondary structure prediction method for single RNA is the Zuker algorithm (Zuker et al., 1989) which looks for the fold with the lowest free energy out of all possible folds. It is an energy minimisation algorithm which assumes that the correct structure is the one with the lowest equilibrium free energy (Durbin et al., 1998).

RNA secondary structure prediction using comparative methods is generally considered to be the most reliable means of determining the RNA secondary structure. The folds of structural RNAs are highly conserved among all kingdoms of life and have been widely used to determine phylogenic relationships between different species (Kumar and Rzhetsky, 1996). The phylogenic-comparative analysis can be used to predict these folds. Most algorithms of this type rely on an analysis of aligned nucleotide sequences to determine conserved regions of secondary structure.

A major problem with early comparative sequence analysis algorithms was that phylogenic relationships of the aligned sequences and levels of sequence divergence were not considered (Silverman, 2003). Inferring the correct structure here requires knowing a structurally correct multiple alignments, but inferring correct alignments requires knowing the correct structure. A structure is solved by an iterative refinement process of guessing the structure based on the current best guess of the multiple alignment, and then realigning based on the new guess at the structure (Gardner and Giegerich, 2004).

### **A new dimension to the Problem**

Although both these methods perform similar functions, biologically the interpretation of their object function is quite complex. The object function provides a criterion of evaluating the system parameters against the function goal. Energy minimization is readily physically understandable, although parameters can be hard to determine since the energy function is simplified and normally not based on physical chemistry but empirical fitting to a coarser model. The object function in the comparative method is derived from observing evolution.

Recently it has been possible to image RNA molecules on a plate, using the technology of Atomic Force Microscopy (AFM) which is elaborated more in chapter 2. It is a precise method, which is capable of achieving a sub-nanometre resolution of images (Renko, 2004).

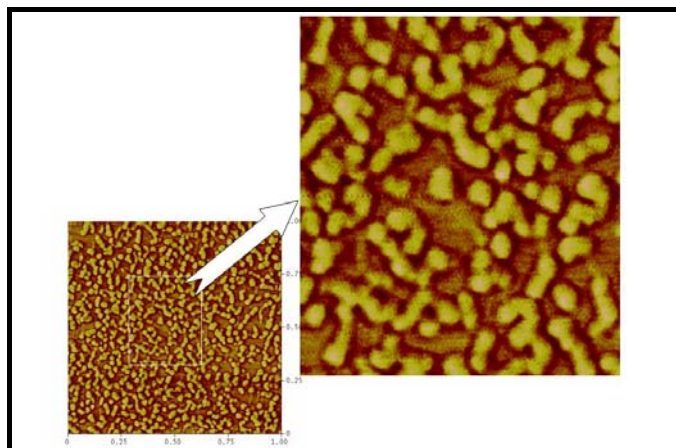


Figure 2 - A sample picture of HIV RNA of 744 nucleotides

*Figure 2* shows a sample picture of HIV RNA of 744 nucleotides that has been refolded in Magnesium and plated on a flat spermine surface and *Figure 3* shows one RNA molecule in focus (All AFM images are received from Ebbe Sloth Anderson, Aarhus University). It makes it possible to distinguish single and paired regions, although it does not seem possible to assign nucleotides to these pictures. This is a very direct source of information about the structures in contrast to the approaches mentioned earlier.

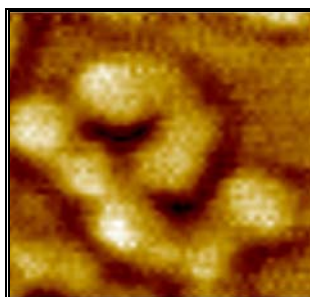


Figure 3 - Zoomed image of a single HIV RNA molecule

An obvious problem now is to annotate such pictures with the known RNA sequences. This would imply describing a distribution on RNA structures obtained from a large set of pictures and then make statements about which positions occupy which part of the photographed structure. An extension to this is combining it with the computational prediction methods and thus using both sources of information. It might seem simple to visualise the process but it is not as obvious when it comes to the extraction of information from these images.

The project will start by focusing on one such image of a particular structure and then try analysing it in comparison with its corresponding molecular structure containing all the nucleotides.

### **Algorithmic Approach**

Many possible secondary structures can be drawn from a sequence. The longer the sequence, the higher the number of possible structures and thus more important it is to distinguish between the structures which are biologically correct and those that are not (Durbin et al., 1998). Hence, there is a need of a ranking function which ranks the correct structure as the highest. We then also would require an algorithm to calculate the score of each possible structure.

The project will start with an implementation of the Nussinov algorithm (Nussinov et al., 1978) to get a better understanding of the structure prediction process. This is because the mechanics of a Nussinov algorithm are the same as those of the most sophisticated energy minimisation folding algorithms. It recursively calculates the best structure for small sub-sequences, and works its way outwards to larger and larger sub-sequences (Durbin et al., 1998). The project will then explore the possibility of implementing other optimisation algorithms suitable for this research.

The ranked set of possible secondary structures of an RNA sequence would allow us to limit the number of possible structures that can map on to the AFM image of the same RNA sequence. Thus, given a list of lengths of paired regions calculated from the measurements carried out on the image, we will try to find the RNA structure that satisfies most of these constraints. One possible approach is to encompass the new information obtained from the images into the prediction algorithm to automatically refine the output, hence ignoring those possible structures that do not for example have the same stem size as that obtained from the image.

*Figure 4* portrays the problem in a diagrammatic form. It shows the image information being input along with the sequence itself into the prediction process, in order to get an optimal structure as output. This will help us verify the correctness of the RNA secondary structure prediction with an assumption that the images portray the correct structure or on the other hand will pose a question to the proclaimed accuracy of the structures captured by the images.

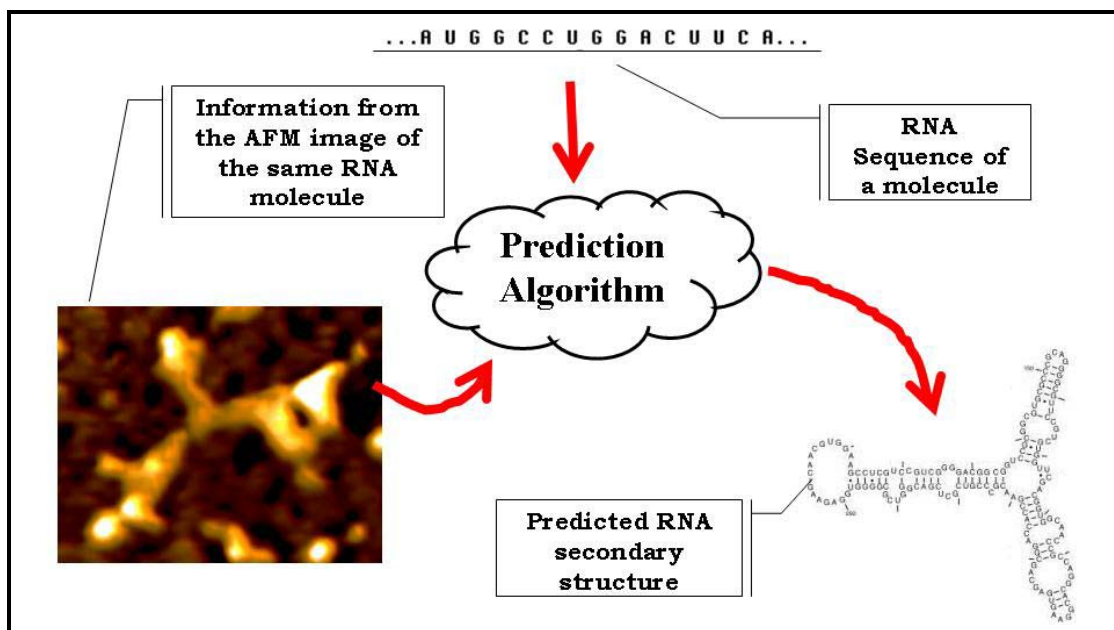


Figure 4 - Pictorial representation of the problem

## Research Aim

The main aim of the project is:

*To analyse avenues utilising the new AFM imaging technology in RNA structure prediction algorithms and explore the idea of these additional constraints leading us to a better folding algorithm.*

Following objectives were set in order to achieve the main goal:

1. A study of various RNA structure prediction algorithms to be carried out in order to explore avenues of how the AFM information can be utilised most effectively.
2. A basic algorithm to be implemented in order to understand the folding procedure.
3. An advanced algorithm to be selected and implemented under the selected AFM constraints.
4. To evaluate the algorithm implemented using test cases simulating information from AFM images.

## *Chapter 2*

# ATOMIC FORCE MICROSCOPY IMAGING OF RNA

### **Introducing Atomic Force Microscopy**

The atomic force microscope (AFM), or scanning force microscope (SFM) was invented in 1986 by Binnig, Quate and Gerber. It was developed as a modification of the scanning tunnelling microscopy. Primarily it was intended for researching conductors and insulators on an atomic scale (Binnig et al. 1986).



Figure 5 - Atomic Force Microscope (Google, 2005)

*Figure 5* shows what an AFM looks like. Like all other scanning probe microscopes, the AFM utilises a sharp probe moving over the surface of a sample in a raster scan. In the case of the AFM, the probe is a tip on the end of a cantilever which bends in response to the force between the tip and the sample. It works by scanning a fine ceramic or semiconductor tip over a surface much the same way as a phonograph needle is used to scan a record (Alonso and Goldmann, 2003). The tip is positioned at the end of a cantilever beam shaped much like a diving board. As the tip is repelled by or attracted to the surface, the cantilever beam deflects. The magnitude of the deflection is captured by a laser that reflects at an oblique angle from the very end of the cantilever. A plot of the laser deflection versus tip position on the sample surface provides the resolution of the hills and valleys that constitute the topography of the surface (Hansama et al., 1997; 2004). The AFM can work with the tip touching the sample (contact mode), or the tip can tap across the surface (tapping mode) much like the cane of a blind person (Ando et al., 2004).

AFM is being used to solve processing and materials problems in a wide range of technologies affecting the electronics, telecommunications, biological, chemical, automotive, aerospace, and energy industries. The materials being investigated include thin and thick film coatings, ceramics, composites, glasses, synthetic and biological membranes, metals, polymers, and semiconductors. The AFM is being applied to studies of phenomena such as abrasion, adhesion, cleaning, corrosion, etching, friction, lubrication, plating, and polishing. The publications related to the AFM are growing speedily since its birth. Some microbiological applications of AFM include Engel and Müller, 2000; Kazuo et al., 2000; Hansama et al., 2003; Edwardson and Henderson, 2004; and Kim et al., 2004.

### **The importance of AFM Information in Bioinformatics**

Recently, a lot of progress has been made in the application of AFM in exploring microbial cell surfaces, membranes, intermolecular/intramolecular interactions and even single molecules i.e. DNA/RNA etc (Renko, 2004). According to Renko, many researchers point out the fact that AFM is currently the only technique that can image the surface of the living cell at high resolution and in real time.

The AFM can probe the differences in shape between many types of biological molecules, such as single-stranded, double-stranded and triple-stranded DNA, and protein channels in membranes. The AFM can also track some biological processes, such as enzymes breaking down DNA, or fibrin clots growing. The AFM tracks these processes even though the molecules are in aqueous solutions and are much smaller than the wavelength of light. Since the AFM is a constantly developing technology, we are just beginning to discover the ways it can be used to investigate biological molecules and biological processes (Hansma, 1997).

### **AFM in RNA Structure Prediction World**

Biological systems can only be fully understood if their structure is known (Aebi et al., 1996). This accentuates the importance of structural biology, the science investigating the structure and function of the components of living systems and it is an area of research where progression relies critically on sophisticated instruments. The AFM is one of the most powerful tools for determining the surface topography of native biomolecules at sub-nanometre resolution (Müller et al., 1995; Zlatanova et al., 2000; Rounsevell et al., 2004). Unlike X-ray crystallography and electron microscopy (EM), the AFM allows biomolecules to be imaged not only under physiological conditions, but also while biological processes are

at work. Because of the high signal-to-noise ratio, the detailed topological information is not restricted to crystalline specimens. Hence single biomolecules without inherent symmetry can be directly monitored in their native environment (Shao et al., 1996; Dufrene et al., 2002; 2003).

RNA molecules are important to study since they are involved in important biochemical functions, including translation, RNA splicing, processing and editing, cellular localization, and catalysis. This also highlights the importance of getting an accurate structure for these molecules. RNA sequence analysis is also different from DNA sequence analysis, since RNA structures fold and base pair with themselves to form secondary structures. Therefore, it is not necessarily the sequence but the structure conservation that is most important in RNA sequence analysis.

There has been a lot of research done and progress made in developing computational techniques to predict accurate structures, as highlighted earlier in chapter 1. However, with the technological development in the area, we can now get a much refined and closer view of an RNA molecule than before. Recent developments also highlight the fact that these images can be obtained in a measurable scale, hence making it possible to keep track of the molecular dimensions.

To the extent of knowing the whole structure from just an image, the technology is still in its early years. Yet it can for sure provide us with valuable pointers that can help us refine our current structure prediction mechanisms and this project aims to highlight this novel concept.

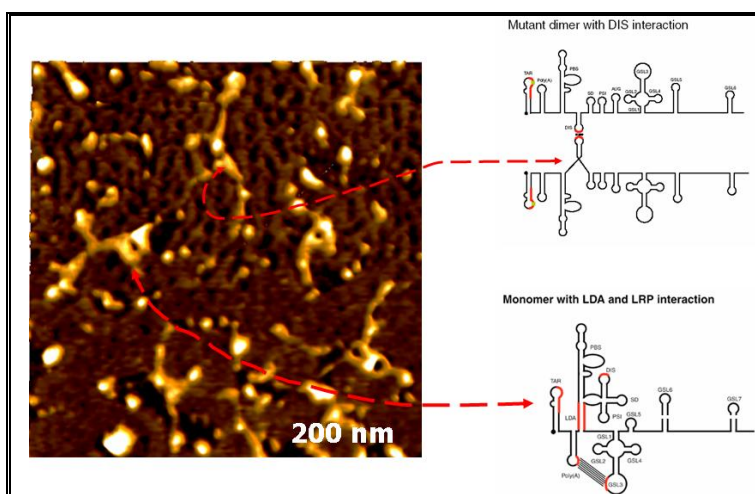


Figure 6 - AFM HIV-RNA image in comparison to the computer generated structures

*Figure 6* shows another AFM image of HIV-RNA molecules (Courtesy of Ebbe Sloth Andersen, Aarhus University). To the right of the image, computer generated structural images of the same molecules are illustrated for comparative analysis. It takes a bit of imagination but by visually comparing the two images, red arrows point to the parts on the images that might be referring to the same section of the molecular structure.

### **The Gulf to Bridge**

As mentioned earlier, certain images were received from the Aarhus University for the purpose of this research. *Figure 6* explained above was the latest image received. Unfortunately, the images received to date are very noisy contrary to earlier expectations; however, researchers in the area are very optimistic. The imaging technology is developing at a great pace and we are not far from getting an immaculate view of an RNA molecule through an improved AFM or a similar much improved technology.

The research started with an aim to process these images for vital information and make this processing a part of the actual structure prediction mechanism. But due to the noise in the images received and the time limitation, the project mainly relied on presumed constraints that an image can possibly reveal and consequently applying them to the RNA structure prediction mechanism.

The project will work on developing a flexible Java based application capable of easily incorporating additional information that we will most probably be able to unveil from such images as technology develops.

## Chapter 3

# EXPLORING RNA FOLDING TECHNIQUES

Research groups across the globe have been working on the determination of RNA secondary structures for decades. The classical approach is the direct observation of a molecule's secondary structure using X-ray crystallography (Golden et al., 1996). Indirect methods involve specific cleavage of the RNA by enzymes called ribonucleases (Aebi et al., 1996). Much of the research has gone into computational prediction of secondary structure from knowledge of the primary structure. The general methods include the search for configurations of maximum base-pairing or of minimum free energy and there has also been research in using comparative analysis methods (Gardner and Giegerich, 2004) mentioned earlier in chapter 1.

This chapter will show a discussion of the present RNA structure prediction approaches and will also explore some search and optimisation algorithms introduced in the MSc Computer Science course modules Intelligent Systems I and II, before deciding on a particular approach for this project.

Before exploring its prediction techniques, it is important to have a mathematical understanding of the RNA structure itself.

### A Mathematical Model of the RNA Secondary Structure

Aebi et al. (1996) and many other researchers have described the RNA secondary structure in a mathematical form. For the ease of understanding the RNA molecule sequence can be represented as a *sequence*  $S$  of symbols:  $s_1, s_2, \dots, s_n$ , where  $s_i$  is one of the nucleotide symbols G, C, A or U. A subsequence of  $S$  may be referred to as a *sequence* itself and a sequence or subsequence may also be called a *string*.

Given a sequence  $S$ , the secondary structure of  $S$  can be represented by the upper right triangular sub-matrix of an  $n$ -by- $n$  matrix  $M$ .  $M_{ij}$  is 1 if  $pair(i, j)$  is true, i.e., for  $i < j$ , if the bases at positions  $i$  and  $j$  in the sequence are paired (Watson-Crick pairing), and is 0 if they are not paired. The explanation of the Watson-Crick will follow later in this section.

The secondary structure may then also be represented by a list  $P$  of pairs, where  $(i, j)$  is in  $P$  if and only if  $\text{pair}(i, j)$  is true.

The subsequence from  $s_i \dots s_j$  can be written as  $[i, j]$ . A subsequence is *proper* with respect to a secondary structure  $P$  if, for every paired element in the subsequence, its pair partner is also in the subsequence.

If  $\text{pair}(i, j)$  is a pair and  $i < r < j$  then we say that  $\text{pair}(i, j)$  *surrounds*  $r$ . Similarly  $\text{pair}(i, j)$  surrounds  $\text{pair}(r, s)$  if it surrounds both  $r$  and  $s$ .

Subsequence  $[i, j]$  is *closed* with respect to a structure  $P$  if  $(i, j)$  is in  $P$ . A pair  $\text{pair}(p, q)$  or an element  $r$  in proper string  $[i, j]$  is accessible in  $[i, j]$  if it is not surrounded by any pair in  $[i, j]$  except possibly  $\text{pair}(i, j)$ . It is accessible from  $\text{pair}(i, j)$  if  $i$  and  $j$  are paired.

A *cycle*  $c$  is a set consisting of a closing pair  $\text{pair}(i, j)$  and all pairs  $\text{pair}(p, q)$  and unpaired elements  $r$  accessible to it.

RNA secondary structures are formed under two types of constraints; hard and soft constraints. Hard constraints put a restriction on certain kinds of pairings and determine if a structure is legal or not, whereas soft constraints are imposed by thermodynamics upon the classes of possible structures and help in determining the optimal structure.

Following are the hard constraints:

- *Watson-Crick pairing*: If  $P$  contains  $(i, j)$  then  $s_i$  and  $s_j$  are either  $G$  and  $C$ , or  $C$  and  $G$ , or  $A$  and  $U$ , or  $U$  and  $A$  or in some cases the rare  $GU$  pairings.
- There is no overlap of pairs. If  $P$  contains  $(i, j)$ , then it cannot contain  $(i, k)$  if  $k \neq j$  or  $(k, j)$  if  $k \neq i$ .
- For all  $i$ ,  $(i, i)$  cannot be in  $P$ .
- Knots are not allowed: If  $b < i < j < k$ , then  $P$  cannot contain both  $(b, j)$  and  $(i, k)$ .
- No sharp loops are allowed: If  $P$  contains  $(i, j)$ , then  $i$  and  $j$  are at least 4 bases apart.

The soft constraint on possible secondary structures  $P$  for  $S$  is simple:  $S$  will assume the secondary structure  $P$  that has minimum free energy. A secondary structure  $P$  for  $S$  can be

described in a natural and unique way as composed of substructures of four kinds: loops, bulges, stacked pairs (a stack of pairs is called a stem), and external single-stranded regions.

If  $P$  contains  $pair(i, j), pair((i + 1), (j - 1)), \dots, pair((i + h), (j - h))$ , each of these pairs (except the last) is said to *stack* on the following pair. Two or more such consecutive pairs is called a *stacked pairs cycle*.

Figure 7 (Aebi et al., 1996) shows examples of these basic structures. According to the notation used in the figure, a pair  $pair(i, j)$  for example is represented as  $i \bullet j$ .

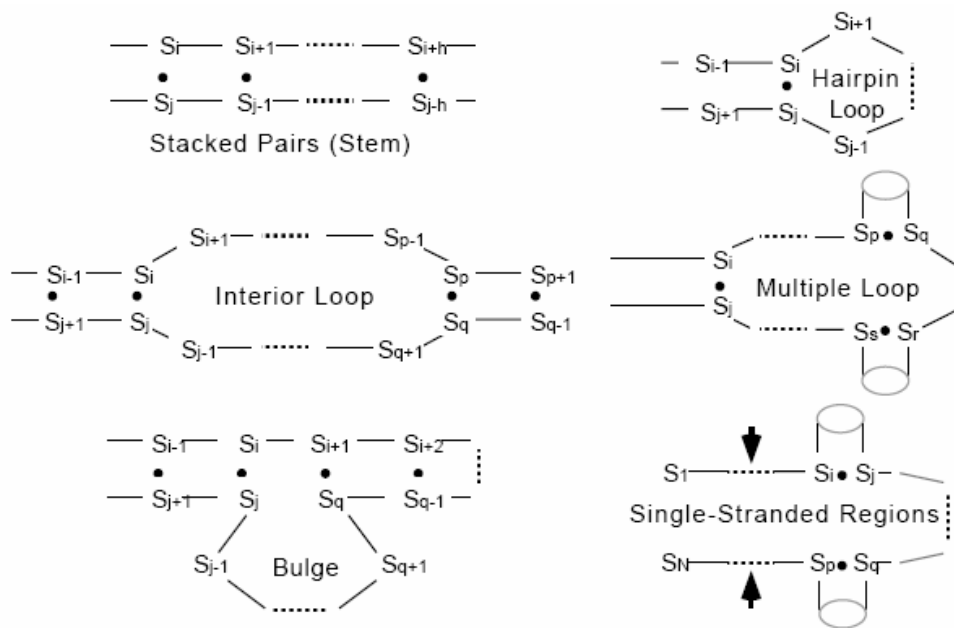


Figure 7 - Basic RNA structures (Aebi et al., 1996)

The soft constraints are as described below:

- If  $P$  contains  $pair(i, j)$  but none of the surrounding elements  $i + 1 \dots j - 1$  are paired, then the cycle is a *hairpin loop*.
- If  $i + 1 < p < q < j - 1$  and  $P$  contains  $pair(i, j)$  and  $pair(p, q)$ , but the elements between  $i$  and  $p$  are unpaired and the elements between  $q$  and  $j$  are unpaired, then the two unpaired regions constitute an *interior loop*.
- If  $P$  contains  $pair(i, j)$  and  $pair(j, i)$  surrounds two or more pairs  $pair(p, q), pair(r, s), \dots$  not surrounding each other, then a *multiple loop* is formed.

- If  $P$  contains  $pair(i, j)$  and  $pair((i + 1), q)$ , and there are some unpaired elements between  $q$  and  $j$ , (or, symmetrically, if  $P$  contains  $pair(i, j)$  and  $pair(p, (j - 1))$  and there are unpaired elements between  $i$  and  $p$ ), then these unpaired elements form a *bulge*.
- Let  $r$  be a sequence of elements in the sequence. If  $r$  is unpaired and there is no pair in  $P$  surrounding  $r$ , then  $r$  falls in a *single-stranded region*.

The Tinoco-Uhlenbeck approach (Tinoco et al., 1971) to specifying the free energy  $E(P)$  of a secondary structure classically rests on the hypothesis that the free energy is a sum of the free energy values of  $P$ 's cycles.

$$E(P) = \sum_i E(c_i)$$

Assuming the equation above still leaves us with the task of specifying free energy values for the primitive substructures, for this we must turn to empirical biochemistry. A lot of research has been done on the problem of assigning free energy values to substructures starting from Tinoco and Uhlenbeck, who made the problem quite simple (Tinoco et al., 1971).

According to Tinoco and Uhlenbeck, it is important to qualify these ideas by noting that the  $E(c)$  free energy estimates for cycles are only estimates. The values cannot be determined with great accuracy, but they serve as useful, if sometimes crude, approximations of physical reality (Aebi et al., 1996). The most stable secondary structures, those having the lowest free energy, are long chains of stacked pairs. That is, a stem is the only kind of cycle which contributes negative free energy to the structure. The particular free energy value for a given stacked pair depends upon the two bases that are bonding, as well as a local context, i.e., the base composition of the closest stacked pairs to its upper right and/or lower left in the matrix.

Loops and bulges raise the free energy roughly in proportion to their size, that is, the number of elements that are left unpaired between the two elements that are paired. Beyond a certain size, loop and bulge energies seem to grow proportionally to the log of the unpaired length. Thus, a certain minimum number of stacked pairs are required to support a loop or bulge interior to the stacked pairs (Aebi et al., 1996).

For the case of simplicity in this research, it can be assumed that the energy is modeled by associating it with all possible structures form the sequence, i.e. stems, bulges and loops etc., and adding it up.

The problem can more formally be defined as follows (Steeg, 1993):

Let  $N$  be a given natural number.

Let  $S$  be the set of RNA sequence of length  $N$ , that is,

$$S = \{A, C, G, U\}^N$$

Let  $P$  be the set of secondary structures for sequence of length  $N$ , that is,

$$P = (0,1)^{N^2}$$

Then, given that  $F$  is a function that assigns free energy values to secondary structures, the search space for the problem is,

Given some  $F$ ,

$$\{(S, P), F(P)\} \text{ where } S \in \Sigma, P \in \Pi \text{ and } F : \Sigma \rightarrow R$$

Where a particular sequence  $S$  is considered and no confusion should result, we can omit the  $S$  argument and use  $F(P)$ .

The problem is thus defined as:

Given a sequence  $S \in \Sigma$ , construct a secondary structure  $P \in \Pi$ , such that:

$$F(P) \leq F(P') \text{ for all } P' \in \Pi.$$

After highlighting the specified problem, the next phase is to explore the possible solutions.

### **RNA Folding as a Search Problem**

Keeping to Watson-Crick base pairing for RNA structure prediction, we can easily see that there are an exponential number of possible structures given a sequence. Even if it is restricted to the chemically possible structures, as defined by the hard constraints mentioned earlier, the number of secondary structures for a given RNA can be unmanageably high

(Durbin et al., 1998). Thus, in order to find the optimal secondary structures, we need to perform a search for the one (or few good structures) among the multitude of very good and bad ones. The problem then is to find ways to restrict the search space and/or to speed up the search through parallelism or by using finely tuned physical parameters to recognize quickly the thermodynamically best structures (Aebi et al., 1996).

### **Exploring Search Algorithms**

Past researchers have applied different kind of search algorithms for RNA structure prediction. Sankoff et al. (1983) describes the application of an exhaustive search algorithm. An exhaustive search algorithm considers every single point in the search space of valid secondary structures for a particular sequence and then calculates the free energy for each of these points, finally selecting the point with the lowest energy value as output. According to Steeg (1993), the algorithm recursively computes the same answer for each subsequence many times and a search algorithm defined in terms of such a recurrence relation is inherently inefficient.

Dynamic programming methods provide a solution by filling in a table of sub-problem values for subsequent repeated lookup, and hence drastically reducing time for the problems which are composed of only a polynomial number of sub-problems (Rivas and Eddy, 1999). However, the complexity concerns for the RNA problem do not derive solely from the choice of full recursion versus dynamic programming. The complexity in the general algorithm stems from the number of possible cycles that have to be considered for each substring if multiple loops of arbitrarily high order are allowed. That is, the number of sub-problems is exponential in the size of  $N$ . This fundamental complexity is not decreased if the algorithm looks up the energy for each possible cycle in a table instead of visiting each cycle substring recursively (Aebi et al., 1996).

An exhaustive search for the best secondary structure is also not a feasible option as some potential structures are not considered due to the restrictive assumptions. It is always a dilemma to keep a balance between which assumptions to make and which classes of structures to ignore. After making some heuristic choices one has to select an approximation algorithm, which will not necessarily provide us with the optimum solution.

The greedy algorithm is another heuristic algorithm which builds solutions incrementally. In a greedy algorithm, at any stage in the building of a solution it is the locally optimal step that is

chosen (Steeg, 1993). A simplistic greedy algorithm for RNA might calculate a structure for an ever larger segment of the RNA sequence. At step  $k$  it would have a secondary structure for  $[1, k - 1]$  and would grow the solution by finding the best way to force the  $k^{\text{th}}$  base onto the current structure. According to Martinez (1984) this although has a biological justification but would generally generate a poor solution, and his method achieved few good results on RNA sequences.

### **Reducing the search space**

As it can be noticed from the discussion that the size of the search space is a crucial component of the search process and one way to move towards an optimum solution is to try to reduce this massive search space. Despite the application of the hard constraints mentioned earlier in the chapter the search space remains considerably huge. This is where the utilization of the new AFM technology could prove to be really handy. The additional information extracted from the images can drastically reduce the search space, possibly providing us with better/signposted pathway to the global optimum solution.

### **Early history of RNA Structure Prediction**

In 1975, Pipas et al. designed an algorithm that performed a three phased search. The first step constructed a list of all possible stems of a certain size. In the second phase this list was then scanned for regions that did not form knots and shared no bases in common. Finally, the algorithm performed an exhaustive search for the set of compatible stacking regions with the lowest free energy using the Tinoco rules (Tinoco et al., 1971). Pipas et al. algorithm was later developed to the Studnicka algorithm (Studnicka et al., 1978).

Dynamic programming for RNA Structure prediction was introduced with the Nussinov algorithm (Nussinov et al., 1978). The Nussinov group published a method to find the configuration with the greatest number of paired bases. The Nussinov algorithm initially aimed to find the configuration with the maximum number of base pairs ignoring the effect of loops. The developed version (Nussinov et al., 1980) then imposed a simple linear penalty on loop size.

### **Application of Intelligent Systems**

There is a growing interest in the application of artificial intelligence (AI) techniques in bioinformatics. Some examples of AI application are gene expression analysis and

protein/RNA folding etc. The AI approaches were mostly applied to areas of *weak constraints*, i.e. where there is no black and white answer and there is always an attempt for a better-than-present option (Russell and Norvig, 2003). Thus the weakly constrained nature of bioinformatics problems makes AI application very suitable. The nature of the problem of RNA structure prediction, where we strive for the best possible structure given a sequence of nucleotides, makes it a suitable application for AI.

### **Machine learning techniques**

According to Oliver G. Selfridge (1993) “If an expert system--brilliantly designed, engineered and implemented--cannot learn not to repeat its mistakes, it is not as intelligent as a worm or a sea anemone or a kitten.”

Machine Learning is generally taken to encompass automatic computing procedures based on logical or binary operations that learn a task from a series of examples (Michie et al., 1995; Baldi and Brunak, 1998). Approaches like *neural networks*, *hidden Markov models*, and *belief networks* etc. are ideally suited for areas where there is a lot of data but little theory and molecular biology is an ideal candidate.

Generally, there are two types of learning schemes in machine learning: *supervised learning* where the learner has some prior knowledge of the data; and *unsupervised learning* where no prior information is given to the learner regarding the data or the output (Russell and Norvig, 2003).

Since the introduction of machine learning to this field, various algorithms and methods have been produced and applied to study different data sets. Most of these studies compare new algorithms with the conventional ones, asserting the effectiveness and efficiencies of their methods in particular data sets or situations (Tan and Gilbert, 2003).

### **Artificial Neural Networks**

The concept of artificial neuron was developed hoping to achieve human-like performance from machines. It emulates the working of a neuron in the biological brain.

Artificial neural networks (ANN) are made up of layers of processing units connected to each other using weights. Each unit computes a weighted sum of its inputs, assumes a new level of activation, and sends an output signal to the units it is connected to. The connections

are variable in nature and determine the strength of the activation which is passed from one unit to the next. The neural network learns data through the modification of these weights. A method known as back-propagation is used in supervised learning where the output from the neural network is compared with the desired output from the data and the error from this is used to change the weights to minimise the error.

AI literature has identified several advantages of neural networks. They can perform with better accuracy than equivalent symbolic techniques on the same data. ANNs have been applied with some success to optimization problems and function approximation. ANN application to the RNA structure prediction problem has also had successful results in the past (Narayanan, 2002; Elrod and Stansfield, 2002, Jeong et al., 2003).

The RNA structure prediction is a complex problem where the set of solutions is huge and not necessarily optimal and many more or less equivalent solutions have variable values in common. It is also an inherently parallel problem and may also require the processing of very noisy or incomplete input data, and one would still like a reasonable answer. Although a neural net architecture meets these requirements, there are certain strings attached. Neural networks are ideal for situations where there is a lot of data available and data also often has to be pre-processed to conform to the requirements of the input nodes of ANNs e.g. it must be normalised and converted into binary form etc. Secondly, it needs training and training first of all needs data/experiments and secondly training times can be very long in comparison with symbolic techniques. Finally, and perhaps most importantly, solutions are encoded in the weights and therefore are not as immediately obvious as the rules and trees produced by some symbolic approaches (Narayanan et al., 2002).

### **Genetic Algorithms**

A Genetic Algorithm is an evolutionary computing algorithm that follows the Darwinian principal of 'survival of the fittest' and was invented by John Holland (1975) and his subordinates. Rechenberg introduced the idea of evolutionary computing in the 1960s in his work "Evolution strategies" (Goldberg, 1989).

Evolutionary computation was based on the idea of using an (artificial) evolutionary process in order to evolve solutions to complex optimization problems. An EA works by creating a population of candidate solutions, testing the quality of each of these candidate solutions, and consequently creating a new pool of candidate solutions by randomly mixing properties from

the best candidate solutions from the current population (Narayanan et al., 2002; Deschênes et al., 2004).

In 1992, Koza used GAs to evolve programs to perform certain tasks. He called his method Genetic Programming. These search algorithms based on the mechanics of natural selection and natural genetics, are particularly suitable for solving complex optimisation problems and for applications that require adaptive problem solving strategies. GAs provide a robust search technique and are particularly suitable for solving problems for applications that require adaptive problem solving strategies (Negnevitsky, 2002).

The algorithm works by selecting *parent* solutions from which to create new *offspring* solutions for the next generation. We want fitter parents to get lots of offspring and parents of lower fitness to get fewer offspring. A basic standard method to create this bias is roulette wheel selection, which can be described as using a virtual roulette wheel with the same number of slots as the population size.

After selection of parents, new candidates can be generated by applying genetic variation using crossover and mutation operators. One example of using the crossover operator is cutting the two parent strings at the same randomly chosen position and creating two offspring by splicing together the left part of the first parent with the right part of the second parent, and vice versa. Mutation is applied by randomly changing chosen characters in the two offsprings to new characters which are also randomly selected.

Since an RNA structure is largely formed by base pairs such as C-G, A-U, and may be G-U, an RNA sequence of  $n$  bases will have approximately  $2^{n/2}$  possible folds (Durbin et al., 1998). Algorithms for prediction of RNA folding therefore either rely on heuristics to prune away large parts of the search space, or on the use of stochastic search algorithms like genetic algorithms in order to quickly locate promising folds. Wiese et al. (2005) in a recent research compared the accuracy of an Evolutionary Algorithm with a Dynamic Programming Algorithm, and the Evolutionary Algorithm clearly outperformed the other in all variable sized testing sequences. GA application in the area has been under research for over a decade and some main contributions include Gulyaev et al., 1995; Shapiro and Wu, 1996, 1997; Chen et al., 2000; Shapiro et al., 2001 and Lee and Han, 2002.

## Selecting an Algorithm for the Project

Before selecting the algorithm it was important to keep the main purpose of the project in mind, which was to see how additional information from techniques like Atomic Force Microscopy can be utilised in developing existing RNA structure prediction techniques. Hence, a basic, easily expandable, data independent and efficient algorithm was desired. Main candidates emerging from the discussion included Neural Networks and Genetic Algorithms. The Nussinov algorithm being one of the pioneers in the field could also provide an excellent tool to help understand basic concepts of the problem itself.

Genetic Algorithm was finally selected for this project. Its data independent nature gave it an edge over Neural Networks and also the fact that further research in this area would have been an expansion to my then-current knowledge base of intelligent techniques introduced in the MSc course. Nussinov Algorithm was also selected for implementation, and was to be used as tool to understand the basic concepts before developing the GA.

A post implementation concern was the evaluation of the developed application. Although there are several online tools like MFOLD (2005) available for RNA structure prediction to test the validity of the developed algorithm, it was difficult to find an evaluation strategy to evaluate it after applying the additional constraints. Hence, a self-orchestrated evaluation strategy was employed for the purpose as explained later in chapter 5.

## Nussinov Algorithm for RNA Folding

The mechanics of a Nussinov algorithm are the same as those of the most sophisticated energy minimisation folding algorithms. It recursively calculates the best structure for small subsequences, and works its way outwards to larger and larger subsequences (Nussinov et al., 1978; 1980).

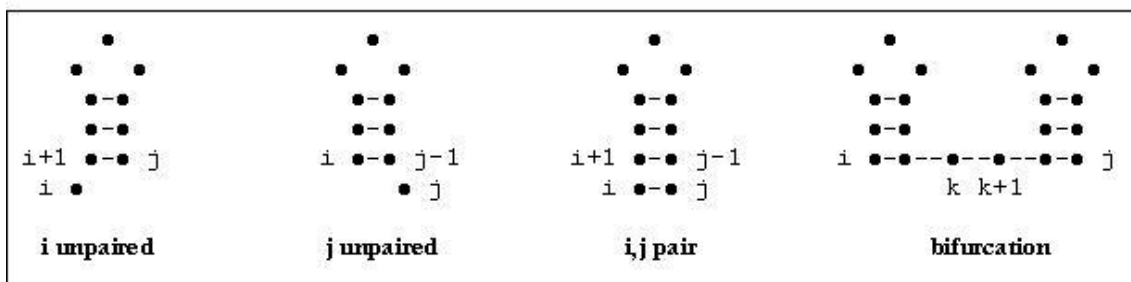


Figure 8 - Nussinov ways to extend an optimal substructure

It is mainly based on four possible ways to extend an optimal substructure as shown in *Figure 8*.

- a. Add an unpaired base  $i$  to the best structure for the subsequence  $i+1, j$
- b. Add an unpaired base  $j$  to the best structure for the subsequence  $i, j-1$
- c. Add paired bases  $i-j$  to the best structure for the subsequence  $i+1, j-1$
- d. Combine two optimal substructures  $i, k$  and  $k+1, j$

The complete algorithm can be found in *Appendix A*.

### **Genetic Algorithm for RNA Folding**

A genetic algorithm starts by creating a pool of all possible pairs and consequent stems that can be generated from the given sequence. Since the stem pool can be very large, hard and soft constraints described earlier in the chapter are used not only to mould the sequence in to a structure but also to reduce the search space considerably. Non-overlapping stems are then randomly picked, hence deriving an RNA structure by moulding the non-paired elements into loops.

From this pool of structures, random selection is used again to pick up an initial population of structures for the starting generation. Each individual in this population is then evaluated by deriving the fitness/energy of the structure by applying a set of energy rules. Once every individual is assigned a fitness value, the operators of crossover and mutation come into the picture. To create offspring structures, two parent structures *Parent1* and *Parent2* are first selected. This selection is again based on the fitness of the individual, the fitter the individual, the higher the chances of it getting picked. Then, two incomplete child structures *Child1* and *Child2* are created by randomly selecting a number of stems from the pool, as it was done earlier when creating the initial population. Then additional stems are distributed by crossover from *Parent1* and *Parent2* into *Child1* and *Child2*. Finally, the fitter of the two is selected to be the offspring replacing *Parent1* in the next generation.

Consequently, after each generation run, a fitter overall population is obtained. As the genetic algorithm proceeds, the probability of accepting destabilizing stems is gradually minimised resulting in the algorithm eventually converging to a population with very little diversity, and this is when the run can be terminated and best structure of this population is expected to be the best possible structure of the particular sequence.



an analysis of several RNA prediction algorithms which were discussed earlier in chapter 3. The requirements were fine tuned throughout the course of the project based on the availability of resources.

The system development phase had two sub-phases. The area is highlighted by the red dotted line around it (Figure 9). A Nussinov algorithm based RNA structure prediction program was developed in the first phase and the second phase included development of the main GA based tool, 'GARNAFold'.

### Implementing Nussinov

To fully understand the process of RNA structure prediction, the implementation of Nussinov algorithm proved to be extremely useful. The Nussinov algorithm based RNA structure prediction algorithm was written in an Object Oriented manner using Java. The choice of the language and the structure was mostly based on the previous programming experience in Java and secondly for the purpose of easy integration with other developments later on in the project.

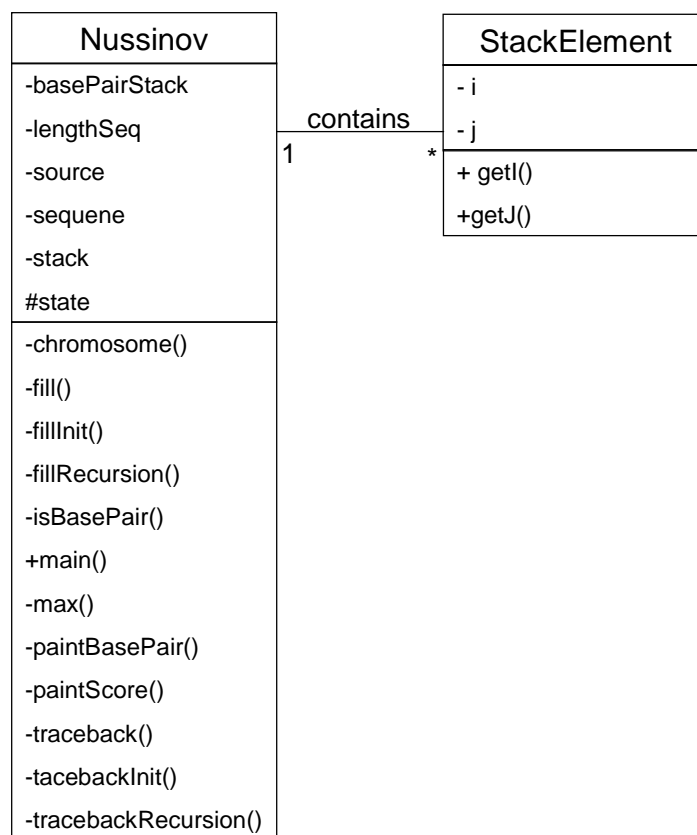


Figure 10 - Nussinov Class diagram

### ***Design details***

The algorithm required two classes; Nussinov and StackElement. Nussinov class is the main class implementing Nussinov algorithm, and StackElement class is only used to create a class object of a pair  $(i, j)$  to be stored in the ‘stack’ of the Nussinov class. The methods and attributes of the classes and their interaction are shown in *Figure 10*.

The main methods of the Nussinov class include Chromosome, Fill, FillInit, fillRecursion and isBasePair. The complete commented source code in *Appendix C* describes all these methods. The Nussinov algorithm itself is described in *Appendix A*.

### ***Implementation details***

One important issue was to decide on the representation of the structure. Nussinov Algorithm relies on an n-by-n matrix representation as explained earlier in chapter 3. This was implemented using ‘score’, a two dimensional double array covering the length of the sequence in both dimensions. So, for example to represent a base pair A-U, where A is at index 3 of the sequence and U at 9, we would simply assign a value of ‘1’ to the coordinates (3, 9) of the two dimensional array. This was implemented using methods including chromosome, fill and fillInit (see *Appendix C*).

As explained earlier in chapter 3, Nussinov aims towards maximising the number of base pairs. Hence, the next important step was to score the structure based on this criteria. *Scoring* was also done on the same representation in order to find the maximum number of base pairs. The values are incremented of the base pairs that are more likely to meet the goal and eventually by tracing these values backwards in the same matrix we can get to the final sequence.

### **Implementing GARNAFold**

GARNAFold is a GA based RNA folding algorithm coded in Java for the purpose of this research. The application consists of 3 main coded classes. A few supporting classes were also used from a package called JGAP (2005), a Java based genetic algorithm package. The object oriented code of GARNAFold can be found in *Appendix D*.

## Design Details

The UML class diagram in *Figure 11* show the classes, their methods, attributes and associations.

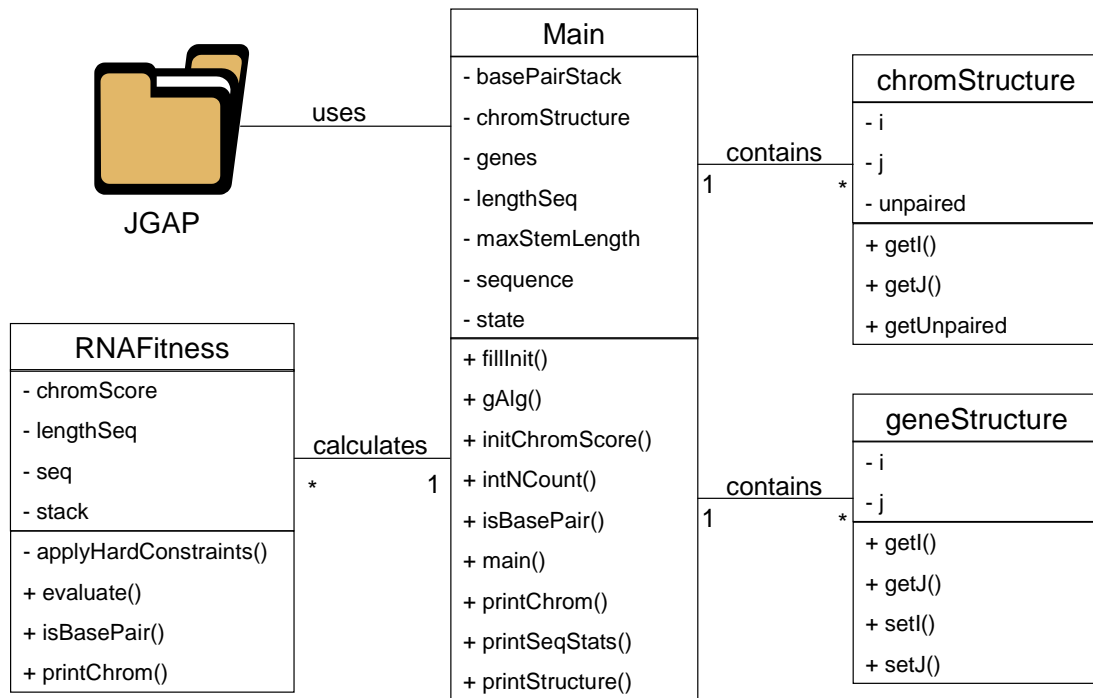


Figure 11- Class diagram for GARNAFold

It highlights the relationship between the four main classes and the off-the-shelf classes used from the package JGAP.

The following section describes a few important stages in the creation of this genetic algorithm based application.

## Implementation details

The *Figure 12* explains the algorithm in a pictorial form. The main stages numbered in the figure are as follows:

1. A search space is created from the input RNA sequence.
2. A specified number of chromosomes (possible RNA structures) are selected to form a population (a set of possible structures). The number of chromosomes in a

population and the size of the population are definable variables, and depend on the size and type of sequence itself.

3. Each chromosome is evaluated for its fitness using a fitness function.

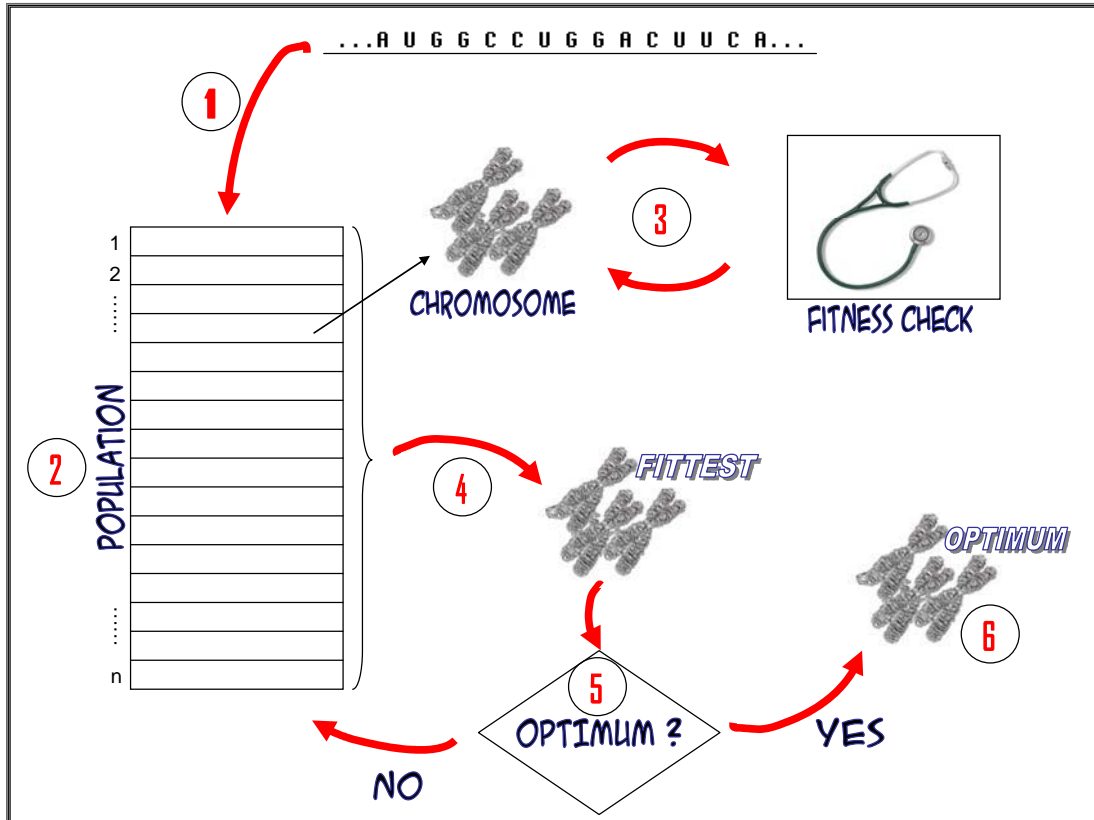


Figure 12 - Genetic Algorithm of GARNAFold

4. The fittest chromosome is selected from the population.
5. This chromosome is evaluated for being optimum. If it is not, then we return to step 2, where an evolved population based on the fit individuals from the last population replaces the current population.
6. If it is the optimum chromosome, it is selected as the output structure for the algorithm output.

In order to implement these phases a few important issue had to be dealt with.

## ***Planning a Chromosome***

The first main concern was to create a chromosome, the most central object in a genetic algorithm. The chromosome represents a potential solution and is further divided into multiple genes. Recalling from chapter 3, genes represent distinct aspects of the solution as a whole, just as human genes represent distinct aspects of individual people, such as their sex or eye colour. In GARNAFold a chromosome was to represent a possible RNA secondary structure and consequently genes were to represent the different base pairings or non paired nucleotides in this chromosome.

During the evolution process, chromosomes are exposed to multiple *genetic operators* (see chapter 3) that represent *crossover*, *mutation*, etc. and then are chosen for the next generation during a natural selection phase based on their *fitness*, which is a measure of how optimal that solution is relative to other potential solutions. The use of genetic operators mentioned above was fairly automated in this case due to supporting JGAP classes used.

Method `gAlg()` in the Main class deals with the creation of the chromosome by first defining individual genes. The same method is then used to set the default configuration of the genetic algorithm, initialise its population and also fitness calculation (*Appendix D* has the detailed GARNAFold code).

## **The Representation Issue**

Another important decision was to decide on the structure representation of the chromosomes described above. This includes the number of genes to be used and their respective alleles (values). The n-by-n dimensional structure as used in the Nussinov algorithm implementation was a good option, however, for the purpose of easy population generation, fitness evaluation and consequently generation evolution, a one dimensional structure was thought to be more suitable. To accomplish the transition a method was used to transform the two dimensional representation into a one dimensional form by utilising the index of the one dimensional structure as another virtual dimension on its own. For example, a value of '1' at coordinates (3, 9) of the two dimensional structure would mean that the 3<sup>rd</sup> and the 9<sup>th</sup> nucleotides form a base pair. In the new one dimensional representation, the same would be represented by a value of 9 at index 3.

### ***Choosing a Fitness Function***

Creating a fitness function was the most important stage in the process. It is a single class method that accepts a potential problem solution, in this case a potential RNA secondary structure and returns an integer value that indicates how good, in other words *fit* that structure in relation to other possible structure.

Recalling from chapter 3 that using the energy minimisation fitness evaluation strategy the lower the value the fitter it is, but for simplicity/compatibility purposes the application is designed with the-higher-the-number, the-better-the-structure concept and similarly the lower it is (only numbers greater than zero being legal fitness values) the poorer the solution.

These fitness measurements are then used to evolve the population of solutions toward a more optimal set of solutions. Since the goal of our algorithm is to produce an optimum secondary structure, the creation of the fitness function evolved around the criteria determining what an optimal structure is. In other words, the hard and soft constraints also described earlier in chapter 3 were used to gauge the fitness of a chromosome. As there are no set priorities of which constraint contributes more towards making a structure optimal, a trial and error based approach was adopted to derive the ideal set of parameters for additions/subtractions to the fitness of a chromosome.

Each one of the hard constraints was part of the evaluation criteria for the fitness evaluation. The chromosomes were scored for meeting any constraint. Negative scoring was also used to discourage structure moulding to an undesired shape or size.

The method `gAlg()` in the Main Class was again used to call the fitness evaluation class. The `RNAFitness` class was used for this purpose. The main method dealing with evaluation of the chromosomes is the `evaluate()` method of the `RNAFitness` class, which calls the method `applyHardConstraints()` of the same class in order to evaluate a chromosome on each individual constraint (*Appendix D* has the detailed `GARNAFold` code).

### ***Creation and Evolution of the Population***

To start the process an important step was the creation of an initial population of sample chromosomes. A method of generating a population of chromosomes in other words a Genotype was the next step. Method `gAlg()` was used again for this purpose. Depending on the size we need for the initial population, individually generated chromosomes are passed

into the initial population. A configuration object is responsible for this process and then this initial population with the correct number of chromosomes, each of which has its genes randomly picked from a defined pool of possible value marks the start of the process. In other words, this random population is the first one to be evaluated, individually fitness-checking each member of this population.

The next step is to evolve this population until it contains some potential solutions that we are satisfied with. Fitness being this satisfaction criteria, takes offspring of fitter individuals forward in to the next generation. Typically, after each evolution cycle, it is important to check if the population contains any satisfactory solutions. After a number of evolutions normally a best structure or a few best structures start dominating every generation. This is when the process can be stopped giving us the resulting optimal structure.

### **Using Information from the AFM Images**

To compensate for the lack of images certain supposed constraints were used, for example stem length and loop size etc were introduced to the algorithm. Constraints were programmed to directly impact the fitness evaluation function, considerably manipulating the fitness value of a chromosome meeting/contradicting the requirements set out by these constraints. The application was designed to take these constraints as input to the algorithm. To show the importance of these constraints making a difference to the prediction process, the stem length constraint is the one selected to be stressed upon more on due to its ease in implementation and also availability of test cases to test its effectiveness.

## *Chapter 5*

### **EVALUATING THE ALGORITHMS**

After the implementation of the algorithms, next step was to evaluate them on different data sets. This chapter explores the behaviour of these algorithms and gauge their effectiveness/success in accordance with the main goals of the research. Due to the novelty of the mechanism, and consequently lack of relevant data sets, a self orchestrated evaluation strategy was employed.

#### **Evaluation Strategy**

The main input to the prediction algorithms is a valid RNA sequence. Several variable length sequences were selected for testing and evaluation purposes. Each one of them provided us with a test case.

The first phase of the testing process was composed of folding these sequences into appropriate structures under normal circumstances using Nussinov and the GARNAFold algorithms. Due to Nussinov being a fairly basic algorithm which does not consider all hard level constraints, the MFOLD server (2005) was also employed to double check the folding mechanism of these algorithms.

Recalling the main goal of the research, next important stage was to test the ability or effectiveness of utilising some additional constraints in the prediction process. Thus, in this phase the prediction process was repeated with additional dimensional constraints, hence testing the ability to restrict the predicted structure to defined constraint. This was to test that knowing the actual length-to-be of the RNA Structure, we should be able to restrict our search space, consequently compelling the prediction process to only consider base paired stack(s) of that specific length.

As mentioned earlier, the much needed sample AFM images were not available during the course of this project, hence, presumed constraints that could possibly be obtained from such an image were employed for the testing and evaluation of these algorithms.

## Selecting appropriate Test Cases

The selection of appropriate test sequences was an important process. As this is an early phase of research in this area, it was considered appropriate to test the system with smaller sequences and fewer constraints. Several sequences of variable lengths ranging from 10 to 60 nucleotides were considered and the main constraint stressed upon was the ‘stem length’.

A list of sequences used is shown in *Appendix B*.

## Carrying out the Tests

Several tests were carried out to judge the accuracy and weight of the claim that these constraints do have an important role to play in accurate RNA structure prediction. A few selected tests are shown in this section. The Graphical User interface for the algorithm was not developed hence the images in these test cases are generated manually to display the test results.

Majority of the tests on GARNAFold were conducted with a population size of 100 and the generations were evolved 500 times. Out of the following selected test cases, test 5 was among those tests conducted with a population size of 1000 and where generations were evolved up to 5000 times.

### Test case 1:

A sample sequence ‘GGGAAAUCC’ was used from the Durbin et al. (1998). The sequence is in the book to illustrate the working of Nussinov algorithm. Due to its simplicity, it is also a good example to illustrate the importance of this research.

Both Nussinov and GARNAFold generated the same structure for this sequence as shown in *Figure 13* below. It was also confirmed by the MFOLD server.

**Note:** The blue dots in the images represent ‘pairing’ of the adjacent nucleotides.



Figure 13 - Folded Test Sequence 1 utilising the maximum stem length

*Applying Stem Length Constraint:*

Next part of the test was to apply a stem length constraint to the same sequence. A constraint of 'stem length = 2' was applied. GARNAFold returned the following sequence.

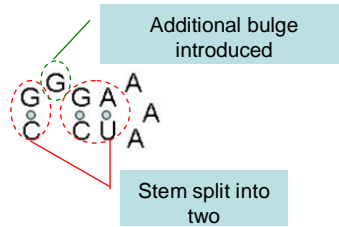
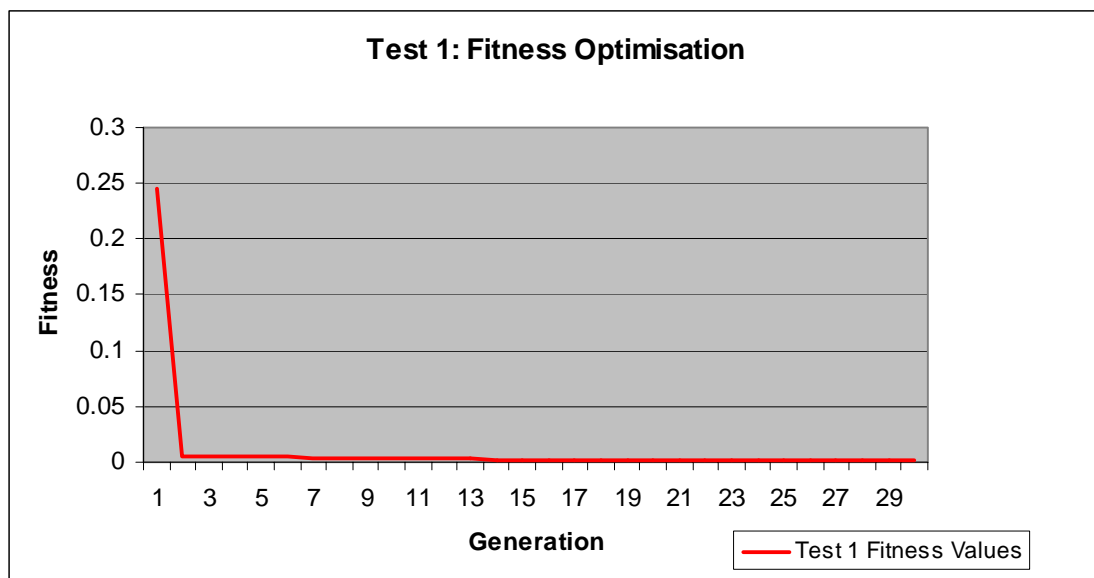


Figure 14 - Test 1 constrained Structure generated by GARNAFold

As it can be noticed that the resulting structure has a maximum stem length of size 2, splitting the stem acquired earlier to two stems of lengths 1 and 2 respectively. This is made possible by algorithmically eliminating all possible stems of size greater than 2 from the search space of all possible structures.

For this small sequence the fitness optimisation was quite fast. The *Graph 1* below plots the normalised values of the fittest chromosomes at every generation and it shows that the optimum structure was found within first 20 evolutionary runs.



Graph 1 - Fitness optimisation for Test case 1

## Test case 2:

Sequence ‘GGCCCGGAAA AACGGUUUCC GGGCC’ was also among the sequences tested. This self created sequence of 25 nucleotides was expected to fold with a longer stem for testing purposes. Following was the resulting structure as generated by Nussinov algorithm.

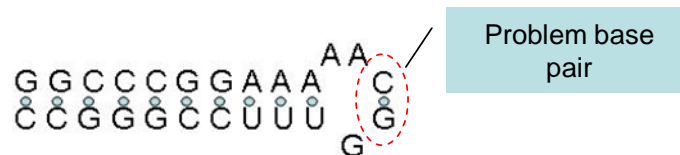


Figure 15 - A flawed structure generated by Nussinov algorithm for test case 2

Nussonov algorithm does not consider the hard constraint of avoiding sharp loops (i.e. the minimum difference of 4 nucleotides between to paired nucleotides) and thus the resulting structure include an extra base pair illustrated on the far right of the image in *Figure 15*. This is not correct according to the generally accepted folding regulations.

GARNAFold is programmed to consider this hard constraint and thus the result shown in the *Figure 16* exclude the extra C-G base pair. MFOLD server also predicted the same structure.

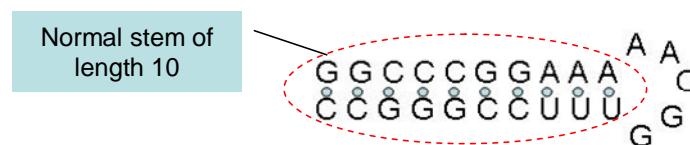


Figure 16 - Test 2 result from GARNAFold and MFOLD

### *Applying Stem Length Constraint:*

The next important step was restricting the stem length. Assuming that the image revealed a maximum stem length of 7 instead of 10 as shown in the predicted structure of *Figure 16*, a revised structure generated with the constraint enabled is shown in *Figure 17* below.

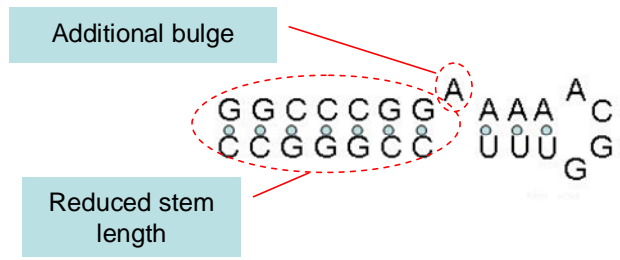
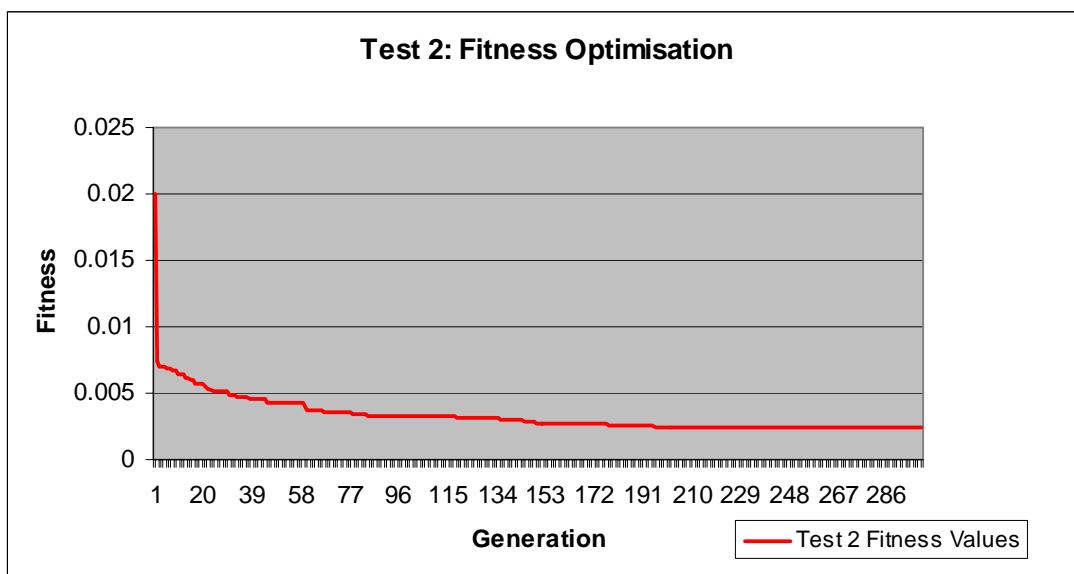


Figure 17 - Test 2 result after applying constraints

The result again was consistent with the findings of previous test. In order to get to a stem of size 7, it forced the stem to split into two, consequently adding an additional bulge and reducing the loop size.

The difference in length of the sequence in Test 2 as compared to the shorter sequence of Test 1, also had an impact on the convergence to the final solution. The *Graph 2* below illustrate that the fitness optimisation in this sequence took place slightly later down the evolutionary process at almost around 200<sup>th</sup> generation. This shows how a slight difference in sequence length can have an impact on the search space by drastically increasing it in size and consequently escalating the time to reach the optimum.



Graph 2 - Fitness Optimisation plot for Test 2 sequence

*Note:* The remaining test cases selected below focus only on illustrating the important constraint based testing of the selected RNA sequences using GARNAFold.

### Test case 3:

Test sequence ‘AAACAUGAGG AUUACCCAUG U’ was also adopted from Durbin et al. (1998). It’s a sequence from the wild type R17 coat protein binding site. The structure shown in *Figure 18* was produced for this sequence by GARNAFold. The MFOLD server confirmed this sequence.

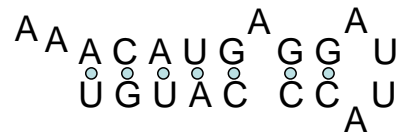


Figure 18 - Test 3 structure from GARNAFold

### *Applying Stem Length Constraint:*

The sequence was tested with a stem length constraint of maximum length 4. As expected the longer stem of length 5 was ruled out of the search space. Following structure was generated as a result and as expected adjustments were made to accommodate the constraint enabled structure (see *Figure 19*).

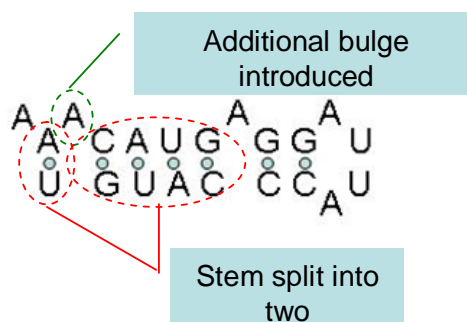


Figure 19 - Test 3 result with a constrained stem of length 4



This scenario illustrates an *ideal application of this research*; a situation, where the AFM imaging technology can contribute a great deal. Supposing that the image of the respective RNA molecule reveal a hint of the actual stem length, it would certainly help us discard the other possible optimum structures that are not matching our known facts.

**Test case 5:**

Last selected test case considered a slightly longer HSA-miRNA-514 sequence acquired from a recent research paper (Bentwich et al., 2005):

‘CUACUCUGGA GAGUGACAAU CAUGUAUAAC UAAAUUUGAU  
UGACACUUCU GUGAGUAG’

The longer sequences need a larger population size and also take a lot many evolutionary runs to converge as the search space expands drastically. Hence, it was difficult to get a convergence with low population and evolution values. These parameters were changed on trial and error bases to get an appropriate structure. A population size of 1000 was eventually used with around 5000 evolutionary runs leading us to an acceptable structure. Such processing also proved to be computationally expensive due to time and space constraints.

GARNAFold ranked a number of structures as optimal. Apart for the hardware constraints mentioned above, it was also due to the ranking system in the fitness function. Energy parameters for fitness function were selected on trial and error basis and its takes time to get optimal parameters.

Following structure was one of the optimal structures produced with GARNAFold and it was also confirmed by MFOLD.

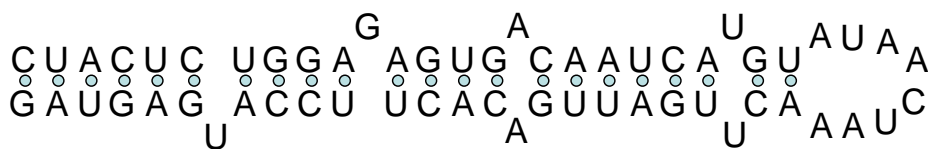


Figure 22 - Test 5 resulting structure

*Applying Stem Length Constraint:*

The structure has about 20 base cases split across 5 stems. It also has small bulges and mini loops all across the stem. One reason for selecting this test case was also to highlight the area

of possible difficulty in this research which needs to be explored even further. From a normal AFM image, it might be difficult to pick up the gaps between the stems e.g. those caused by the one-nucleotide bulges and mini interior loops of 2 unpaired nucleotides. Thus it might be problematic when it comes to determining the exact stem size from the AFM image until or unless the image is crystal clear with no noise.

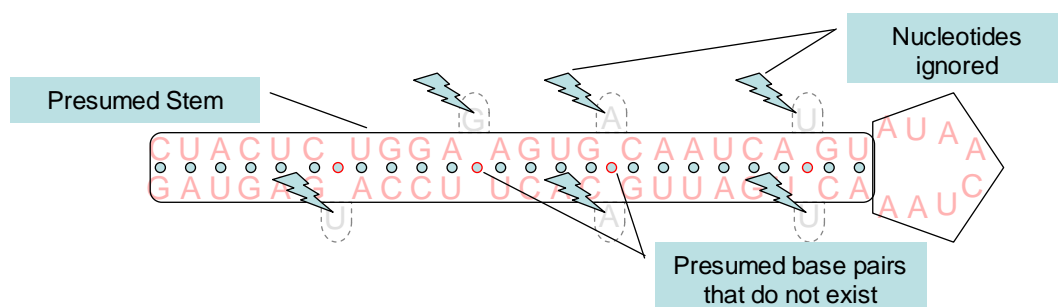


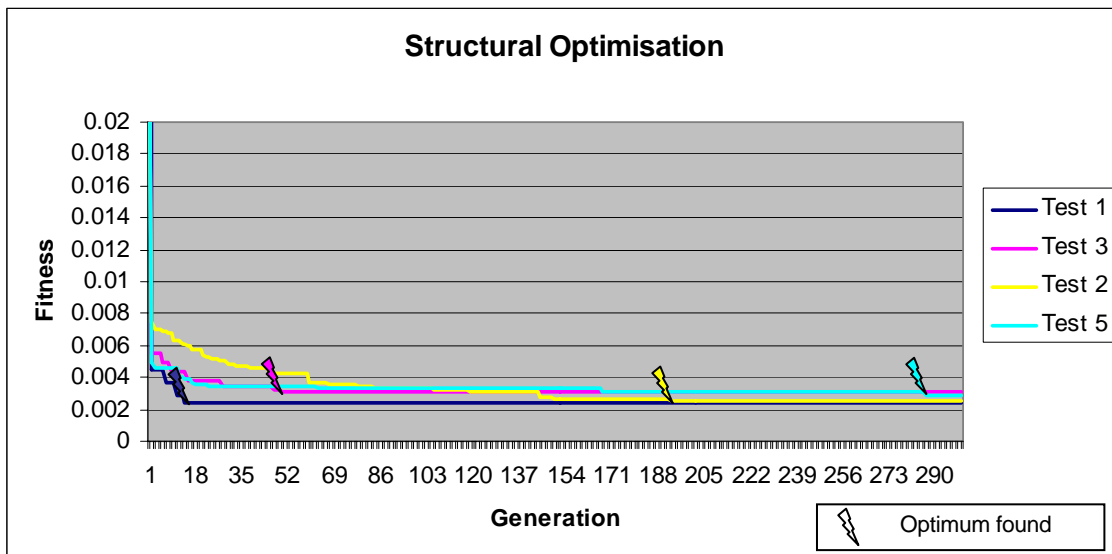
Figure 23 - Presumed AFM image of the RNA structure of Test 5

For testing purposes, it was assumed that the small bulges and loops were not picked up from the image and a stem length constraint of 26 was applied to the prediction process. *Figure 23* shows the presumed AFM image for the structure. According to the assumption the bulges and small loops were rather counted as additional base pairs. When this was applied to GARNAFold, it didn't make any difference at all to the outcome. This was mainly because stem of such length was already ignored by the normal prediction process; hence the constraint had no effect on the process at all producing exactly the same structure as in *Figure 22*.

### Analysis of the results

The results overall show a consistent theme of the constraints being able to influence the moulding of the predicted structures. Given the limited hardware resources, the prediction was more precise for shorter sequences as compared to larger sequences.

*Graph 3* illustrates the comparison of how sequences of variable lengths optimise to a final structure. The lightning arrows point to the generations where optimum structures were found and it shows that as length increases due to a consequent increase in search space, the time taken to the optimum structure also increases.



Graph 3 - Comparison showing the effect of length on optimisation time

Fine tuning of fitness evaluation parameters in the genetic algorithm was the key to improving the quality of prediction and this process itself is based on trial and error.

Based on the parameter used for the testing of 25 sequences, the precision was found to be around 88% when GARNAFold prediction was compared to the MFOLD structure. The remaining 12% i.e. the 3 sequences that gave a different result from MFOLD were composed of over 50 nucleotides in terms of length. The algorithm however was very responsive to the application of constraints with the accuracy of over 90%.

### Issues Encountered

The main issue encountered was the limitation in hardware resources to test larger sequences. This is one drawback of genetic algorithms using a very large search space. Another main concern was the lack of test data to test the algorithm, as no research has been done yet which incorporates additional dimensional constraint information into the prediction process. Generated sample data and presumed constraints simulating AFM image information, proved to be quite an effective substitution in order to help carrying out the tests.

## *Chapter 6*

# CONCLUSIONS

### **Project Review**

The project research started with a couple of sample AFM images of HIV-RNA molecule. It was quite difficult to analyse them due to excessive noise and lack of quality. But one thing was clear that the images were scaled and the technology was constantly improving. Based on these facts it was considered appropriate to assume that certain dimensional constraints are easily obtainable once better images are received.

Due to time limitations, it was decided to start on building a solid foundation for the idea, rather than waiting for more images to arrive. It commenced with a thorough understanding of the RNA structure prediction process through the implementation of Nussinov algorithm. Nussinov is a fairly basic algorithm but it proved to be very useful in clearing up basic concepts.

To test this idea of using AFM imaging information in the prediction process, the next important component of the research was to select an appropriate prediction algorithm. A thorough literature analysis was conducted before selecting Genetic Algorithms for the purpose. The programming language selected was Java mainly due to previous experience in the language.

*GARNAFold*, a genetic algorithm based RNA folding algorithm was implemented. The most important stages in the development of the genetic algorithm were deciding on the appropriate chromosome and gene structure representations and secondly selecting a suitable fitness evaluation function. A one dimensional array structure with the utilisation of its index as a dimension on its own was preferred over an n-by-n matrix representation. This was mostly to speed up the fitness evaluation process for each chromosome i.e. a possible structure, in every population. The fitness evaluation function was custom made to rank every chromosome for its conformation with the hard and soft constraints of RNA structure prediction process. Positive and negative scoring determined the fitness of every chromosome.

Coming to another important phase of the research the next step was the evaluation of the prediction process. Images were still not received; hence it was decided to rather use presumed constraint representing possibly extracted information from such an image. The algorithms were then tested firstly for the normal structure prediction and then most importantly for the utilisation of presumed constraints within the prediction process.

### **Concluding Remarks**

Research developments in the field of AFM imaging has made it possible now to visually analyse an RNA molecule to sub-nanometre scaled resolution. An image of an RNA molecular structure reveals enough information to extract the dimensional measurements of the given structure. These can be of vital significance in reducing the large search space to an acceptable level. Tests carried out repeatedly over a selection of 25 sample mini-RNA sequences proved that this additional information does have a role to play. The application of the presumed stem length constraint to these sequences was consistently able to restrict the folding within the specified stem length constraint, with an accuracy of around 90%.

While testing the genetic algorithm folding, it was also noticed that the prediction quality slightly deteriorated with the increasing length of the sequence. This was mainly due to the exponential search space utilised in the process, the parallel processing ability of genetic algorithms and the limited hardware resources available at the time for the use of suitable genetic operator values. Thus smaller sequences were mostly used and were considered sufficient enough to show the importance of the utilisation of additional constraints.

The evaluation process highlighted the need for improved hardware resources for testing longer sequences and also to explore other suitable prediction algorithms for the purpose. Using the parallel processing nature of a genetic algorithm also has its drawbacks as it uses an enormous search space all at once.

However, the application of the presumed constraints (although at a basic level) to a constraint enabled prediction algorithm which was developed specifically for this research, proved the very point that by utilising some dimensional information about the structure of an RNA molecule (i.e. utilising AFM information), we can considerably improve the accuracy of the prediction process itself.

## **Thinking Forward**

The research is still in its infancy, but has opened up doors for this novel concept to be utilised at a higher level. Further research incentives in this area also rely on the developments in the field of AFM imaging. The image processing techniques can also be employed to automate the information extraction process for more detailed information that can possibly be utilised in the prediction process.

The future work should also focus on testing with a broader set of parameters and constraints on a greater number of sequences of variable lengths and on an improved hardware platform. In addition, as mentioned earlier other algorithms can also be explored for this purpose to check if the application of such constraints can be made even easier.

The research focused on laying down solid foundations for further research in the area and also to give out sufficient enough evidence that the idea is a revolutionary one in this field. To conclude we can say that the future of RNA structure prediction seems fairly bright and we are not far from predicting the structures with near 100% accuracy.

## **MSc Course Relevance to this Project**

This section highlights the relevance of this project to the MSc course itself. The main courses taken that relate to this project include:

- Intelligent Systems I
- Intelligent Systems II
- Requirements Engineering

From the two Intelligent Systems courses, the topics utilised in this research include several problem solving techniques, exhaustive search techniques, greedy search algorithms, dynamic programming algorithms, artificial neural networks and most importantly genetic algorithms. All these were explored in detail and finally a dynamic programming algorithm and a genetic algorithm were selected for the implementation phase.

Requirements Engineering course also proved to be extremely useful in managing the whole project itself. Starting from the problem definition, to the analysis, design and then the implementation phase, several techniques learned were adopted and utilised effectively.

## BIBLIOGRAPHY

### Books:

- Aebi U, Engel A, Müller DJ. *From bones to atoms: Imaging nature across dimensions*. Bauer Druck, Basel, Switzerland, (1996).
- Baldi P, Brunak S. *Bioinformatics: The Machine Learning Approach*. The MIT Press, (1998)
- Durbin R, Eddy S, Krogh A, Mitchison G. *Biological sequence analysis*. Cambridge University Press, Cambridge, England, (1998).
- Elrod S, Stansfield W. *Genetics*. Fourth Edition, McGraw-Hill, NY, (2002).
- Goldberg D. *Genetic Algorithms in searching, optimization and machine learning*. Reading, MA: Addison-Wesley, (1989).
- Holland JH. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. An Arbor, MI: University of Michigan Press, (1975).
- Koza JR. *Genetic Programming: on the programming of computers by means of natural selection*. Cambridge, MA: MIT Press, (1992).
- Michie D, Spiegelhalter DJ, Taylor CC, Campbell J. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood Series in AI, Ellis Horwood, NJ, (1995).
- Negnevitsky M. *Artificial Intelligence*. London: Addison Wesley, (2002).
- Russell S, Norvig P. *Artificial Intelligence: A Modern Approach*. 2nd Edition, Prentice Hall, London.(2003).
- Sankoff D, Kruskal JB, Mainville S, Cedergren RJ. *Fast Algorithms to determine RNA secondary structures containing multiple loops*. In Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison. Addison-Wesley, Reading, MA. (1983).

### Journals:

- Alonso, JL, Goldmann WH. *Feeling forces: Atomic Force Microscopy in cell biology*. Life Sciences, 72:2553-2560, (2003).
- Ando T, Kodera N, Takai E, Maruyama D, Saito K, Toda A. *A high speed atomic force microscope for studying macromolecules*. Proceedings of National Academy of Sciences of the United States of America, 98:12468-12472, (2001).
- Bentwich I, Avniel A, Karov Y, Aharonov R, Gilad S, Barad O, Barzilai A, Einat P, Einav U, Meiri E, Sharon E, Spector Y, Bentwich Z. *Identification of Hundreds of Conserved and Non-conserved Human MicroRNAs*. Nature Genetics, 37(7):766-770, (2005).
- Binnig G, Quate CF, Gerber CF. *Atomic force microscopy*. Physical Review Letters, 56(9):930-3, (1986).
- Chen JH, Le SY, Maizel JV. *Prediction of common secondary structure of RNAs: a genetic algorithm approach*. Nucleic Acid Research, 28(4):991-999, (2000).

- Deschênes A, Wiese KC, Poonian J. *Comparison of Dynamic Programming and Evolutionary Algorithms for RNA Secondary Structure Prediction*. Proceedings of the IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology 214-222, (2004).
- Dufrene YF. *Atomic force microscopy, a powerful tool in microbiology*. Journal of bacteriology, 19:5205-5213, (2002).
- Dufrene YF. *Recent progress in the application of atomic force microscopy imaging and force spectroscopy to microbiology*. Current opinion in microbiology, 6:317-323, (2003).
- Eddy SR. *How do RNA folding algorithms work?* Nature Biotechnology, 22(11):1457-1458, (2004).
- Edwardson JM, Henderson RM. *Atomic force microscopy and drug discovery*. Drug discovery today, 9:64-71, (2004).
- Engel A, Müller DJ. *Observing single biomolecules at work with the atomic force microscope*. Nature structural biology, 7(9):715-718, (2000).
- Evers DJ, Giegerich R. *Reducing the Conformation Space in RNA Structure Prediction*. German Conference in Bioinformatics. Available online: <http://www.bioinfo.de/isb/gcb01/talks/evers/index.html>, (2001)
- Gardner P, Giegerich R. *A Comprehensive comparison of comparative RNA structure prediction approaches*. BMC Bioinformatics, (2004).
- Golden BL, Gooding AR, Podell ER, Cech TR. *X-ray crystallography of large RNAs: heavy-atom derivatives by RNA engineering*. RNA. 2(12):1295-1305, (1996).
- Gulyaev AP, V-Batenburg FHD, Pleij WAC. *The computer simulation of RNA folding pathways using a genetic algorithm*. Journal of Molecular Biology, 250:37-51, (1995).
- Hansma HG, *Probing biomolecules with the atomic force microscope*. Dept of Physics, University of California, Available online: [http://www.physics.ucsb.edu/~hhansma/afm-acb\\_news.htm](http://www.physics.ucsb.edu/~hhansma/afm-acb_news.htm), (1997).
- Hansama HG, Oroudjev E, Baudrey S, Jaeger L. *TectoRNA and 'kissing-loop' RNA: atomic force microscopy of self-assembling RNA structures*. The Royal Microscopical Society, Journal of Microscopy, 212:273-279, (2003).
- Hansama HG, Kasuya K, Oroudjev E. *Atomic force microscopy imaging and pulling of nucleic acids*. Science Direct, Current Opinion in Structural Biology, 14:380-385, (2004).
- Jeong E, Chung IF, Miyano S. *Prediction of Residues in Protein-RNA Interaction Sites by Neural Networks*. Genome Informatics, 14: 506-507, (2003).
- Kazuo U, Nagami F, Okada T, Kuroda R. *AFM characterization of single strand-specific endonuclease activity on linear DNA*. Nucleic Acid Research, 28(9), (2000).
- Kim J, Yoshimura SH, Hizume K, Ohniwa RL, Ishihama A, Takeyasu K. *Fundamental structural units of Escherichia coli nucleoid revealed by atomic force microscopy*. Nucleic Acid Research, 32(6):1982-1992, (2004).
- Kumar S, Rzhetsky A. *Evolutionary Relationships of Eukaryotic Kingdoms*.

- Journal of Molecular Evolution, 42:183-193, (1996).
- Lee D, Han K. *Prediction of RNA Pseudoknots – Comparative study of genetic algorithms*. Genome Informatics, 13:414-415, (2002).
- Müller DJ, Büldt G, Engel A. *Force-induced conformational change of bacteriorhodopsin*. Journal of Molecular Biology. 249, 239-243, (1995).
- Narayanan A, Keedwell EC, Olsson B. *Artificial intelligence techniques for bioinformatics*. Applied Bioinformatics, 1(4):191-222, (2002).
- Nussinov R, Pieczenik G, Griggs JR, Kleitman DJ. *Algorithm for loop matching*. SIAM J. Appl. Math., 35:68-82, (1978).
- Nussinov R, Jacobson AB. *Fast algorithm for predicting the secondary structure of single-stranded RNA*. Proc Natl Acad Sci USA, 77(11):6309–6313, (1980).
- Pipas JM, McMahon JE. *Methods for Predicting RNA Secondary Structures*. In Proceedings of the National Academy of Sciences, U.S.A. 72:2017-2021, (1975).
- Renko M. *Atomic Force Microscopy in Biology*. Seminar, Jozef Stefan International Postgraduate School, Available online at [www.mps.si/ips/seminar.pdf](http://www.mps.si/ips/seminar.pdf), (2004).
- Rivas E, Eddy SR. *A dynamic programming algorithm for RNA structure prediction including pseudoknots*. Journal of Molecular Biology, 5:285(5):2053-68, (1999).
- Rounsevell R, Forman JR, Clarke J. *Atomic force microscopy: mechanical unfolding of proteins*. Methods, 34(1):100-111, Available Online at [www.science-direct.com](http://www.science-direct.com), (2004).
- Selbig J, Mevissen T, Lengauer T. *Decision tree-based formation of consensus protein secondary structure prediction*. Bioinformatics 15(12):1039-1046, (1999).
- Selfridge OG. *The Gardens of Learning: A Vision for AI*. AI Magazine 14(2):36-48 (1993)
- Shao Z, Mou J, Czajkowsky DM, Yang J, Yuan JY. *Biological atomic force microscopy: What is achieved and what is needed*. Advances in Physics 45, 1-86, (1996).
- Shapiro BA, Wu JC. *An annealing mutation operator in genetic algorithms for RNA folding*. Computer Applications in the Biosciences, 12:171-180, (1996).
- Shapiro BA, Wu JC. *Predicting RNA H-Type pseudoknots with the massively parallel genetic algorithm*. Computer Applications in Biosciences, 13:459-471, (1997).
- Shapiro JV, Wu JC, Bangali D, Potts MJ. *The massively parallel genetic algorithm for RNA folding: MIMD implementation and population variation*. Bioinformatics, 17(2):137-148, (2001).
- Silverman A. *A Critical Review of Computational Methods for RNA Secondary Structure Prediction*. Biochem 218, Available online: <http://cmgm.stanford.edu/biochem218/Projects%20Spring%202003/Silverman.pdf>, (2003).
- Steeg EW. *Neural Networks, Adaptive Optimization, and RNA Secondary Structure Prediction*. Artificial Intelligence and Molecular Biology, Lawrence Hunter, ed., American Assoc. Artificial Intelligence, Available online: <http://fano.ics.uci.edu/cites/>

- Document/Neural-networks-adaptive-optimization-and-RNA-secondary-structure-prediction.html, 121-160, (1993).
- Studnicka GM, Rahn GM, Cummings IW, Salser WA. *Computer Methods for Predicting the Secondary Structure of Single-stranded RNA*. Nucleic Acids Research, 5(9):3365-3387, (1978).
- Tan AC, Gilbert D. *An empirical comparison of supervised machine learning techniques in bioinformatics*. Proceedings of the First Asia-Pacific bioinformatics conference on Bioinformatics, 19:219-222, (2003).
- Tinoco I, Bustamante C. *How RNA folds*. Journal of Molecular Biology. 293(2): 271-281, (1999).
- Tinoco I, Uhlenbeck OC, Levine MD. *Estimation of Secondary Structure in Ribonucleic Acids*. Nature (London), 230:362, (1971).
- Wiese KC, Hendriks A, Poonian J. *Algorithms for RNA Folding: a Comparison of Dynamic Programming and Parallel Evolutionary Algorithms*. Proceedings of the IEEE Congress on Evolutionary Computation, Edinburgh, Scotland, Available online: [www.sfu.ca/~jpooniaa/cec2005\\_Hendriks.pdf](http://www.sfu.ca/~jpooniaa/cec2005_Hendriks.pdf), (2005).
- Zlatanova J, Lindsay SM, Leuba SH. *Single molecule force spectroscopy in biology using the atomic force microscope*. Progress in biophysics & molecular biology, 74:37-61, (2000).
- Zuker M. *On finding all suboptimal foldings of an RNA molecule*. Science, 244:48-52, (1989).
- Zuker M, Mathews DH, Turner DH. *Algorithms and Thermodynamics for RNA secondary structure prediction: a practical guide*. A manual produced by The Bioinformatics Center at Rensselaer and Wadsworth, Available online at <http://www.bioinfo.rpi.edu/~zukerm/seqanal/mfold-3.0-manual.pdf>, (1998).
- Zuker M, Stiegler P. *Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information*. Nucleic Acids Research. 9(1):133-148, (1981).
- Websites:**
- Bernhard's Page, *RNA Structure Prediction*. <http://ludwig-un2.unil.ch/~bsondere/nussinov/>, Accessed: 28 July (2005)
- Google, *Google Images: atomic force microscope*. <http://images.google.co.uk/images?svnum=10&hl=en&lr=&q=Atomic+force+microscope>, Accessed: 17 Jun (2005).
- JGAP, *Java Genetic Algorithms Package*. <http://jgap.sourceforge.net/index.html>, Accessed: 23 Jun (2005).
- MFOLD, Prediction of RNA secondary structure (M. Zuker), <http://bioweb.pasteur.fr/seqanal/interfaces/mfold-simple.html>, Accessed: 1 Aug (2005)
- SRPDB. *Halobacterium halobium SRP RNA*. Signal Recognition Particle database, (March 10, 2000) <http://psyche.uthct.edu/dbs/SRPDB/SRPDB.html>, Accessed: 1 Jun (2005)
-

## APPENDIX A – NUSSINOV ALGORITHM

(Algorithm taken from Durbin et al., 1998 and Bernhard, 2005)

### Matrix Fill Stage

*Initialization:*

$$\gamma(i, i-1) = 0 \quad \text{for } i = 2 \text{ to } L$$

$$\gamma(i, i) = 0 \quad \text{for } i = 1 \text{ to } L$$

*Recursion:*

$$\gamma(i, j) = \max \left\{ \begin{array}{l} \gamma(i+1, j) \\ \gamma(i, j-1) \\ \gamma(i+1, j-1) + \delta(i, j) \\ \max_{i < k < j} [\gamma(i, k) + \gamma(k+1, j)] \end{array} \right.$$

## Traceback

The traceback uses a pushdown stack in order to handle bifurcations properly. The stack stores one branch of the bifurcation while the other branch is traced back.

### *Initialization*

Push (1,L) onto the stack

### *Recursion*

Repeat until the stack is empty:

pop(i,j)

if  $i \geq j$  continue

    else if  $\gamma(i+1,j) = \gamma(i,j)$  push (i+1,j)

    else if  $\gamma(i, j-1) = \gamma(i,j)$  push (i,j-1)

    else if  $\gamma(i+1,j-1) = \gamma(i,j)$

        record i,j basepair

        push (i+1,j-1)

    else for  $k=i+1$  to  $j-1$

        if  $\gamma(i, k) + \gamma(k+1,j) = \gamma(i,j)$

            push (k+1,j)

            push(i,k)

            break

## APPENDIX B – TEST SEQUENCES USED

Following are the test sequences used for the evaluation phase :

1. GGGAAAUCC
2. GGCCCGGAAA AACGGUUUCC GGGCC
3. AAACAUGAGG AUUACCCAUG U
4. ACCCCCUCCU UCCUUGGAUC AAGGGGCUCA A
5. CUACUCUGGA GAGUGACAAU CAUGUAUAAC UAAAUUUGAU UGACACUUCU  
GUGAGUAG
6. CAUGUAUAAC UAAAUUUGAU UGAC
7. GACAAU CAUGUAUAAC UA
8. UAAAUUUGAU UGACACUUCU GUGAGUAG
9. CAUGUAUAAC UAAAUUUGAU
10. GAGUGACAAU UGACACUUCU CUACUC
11. CUACUCUGGA GAGUGACAAU CAUGUAUAAC UAAAUUUGAU UGACACUUCU  
GUGAGUAG
12. GUGAGUAG UAAAUUUGAU UGACACUUCU CUACUC
13. UAAAUUUGAU UGACACUUCU ACC
14. GAGUGACAAU CAUGUAUAAU UAAAU
15. GUGAGUAG CAUGUAUAAC CUACUC
16. UAAAUUUGAU ACAAUAGUAG UGACUUUGAU GAU

17. CUACUCUGGA UAGUAGGAUA
18. UUUCAACCCG AAA
19. UAAAUUUGAU UGACACUUCU CUA
20. CAUGUAUAAU GUGAGUAG UGACACUUCU UAA
21. UGACACUUCU GUGAGUAG ACCU
22. CAUGUAUAAC CUACUCUGGA GAGUGACAAU
23. AUAACCCUUG GGAACUUU
24. CUACUCUGGA GAGUGACAAU CAUGUAUAAU UAAAUUUGAU UGACACUUCU  
GUGAGUAG
25. CUACUCUGGA GAGAAAUGAC

## APPENDIX C – NUSSINOV SOURCE CODE

### Nussinov Source code :

Following is the important part of Nussiniv implementation.

```
/*
 * Nussinov.java
 * Created on 01 June 2005, 13:54
 */
package GARNAlgorithm;
/**
 * @author Asim Khan
 * @course MSc Computer Science
 */
import java.io.*;
import java.util.Stack;
import java.util.*;

public class Nussinov
{
    private String sequence = "";
    private int lengthSeq = 0;
    private double[][] score = null;
    private Stack stack = null;
    private Stack basePairStack = new Stack();

    // for stepping through the algorithm
    protected String state = "";

    // Creates a new instance of Nussinov
    public Nussinov(String sequence)
    {
        this.sequence = sequence.toUpperCase();
        this.lengthSeq = sequence.length();
        this.score = new double[lengthSeq][lengthSeq];

        // just for vision
        for (int i = 0; i < lengthSeq; i++)
        {
            for (int j = 0; j < lengthSeq; j++)
            {
                score[i][j] = -1;
            }
        }
    }
}
```

```

//Method isBasePair return 1 if true, else returns 0
private int isBasePair(int i, int j)
{
    char letterI = sequence.charAt(i);
    char letterJ = sequence.charAt(j);

    if (letterI == 'A' && letterJ == 'U' || letterI == 'U'
        && letterJ == 'A' ||
        letterI == 'C' && letterJ == 'G' || letterI == 'G' &&
        letterJ == 'C') return 1;
    else return 0;
}

//Method creates the chromosome
private double chromosome()
{
    double x=0.0;
    for (int i = 0; i < lengthSeq; i++)
    {
        for (int j = 0; j < lengthSeq; j++)
        {
            if (isBasePair(i,j)==1)
            {
                score[i][j]=1;
                for (int previous =j;previous>=0;previous--)
                {
                    if(score[i][j]*Math.random()>score[i]
                        [previous]*Math.random())
                    {
                        score[i][j]=1;
                        score[i][previous]=0;
                    }
                    else
                    {
                        score[i][previous] = 1;
                        score[i][j]=0;
                    }
                }
            }
        }
    }
    return x;
}

private double max(int i, int j)
{
    double a = score[i+1][j];
    double b = score[i][j-1];
}

```

```

    double c = score[i+1][j-1]+isBasePair(i,j);

    double d = 0;
    for (int k = i+1; k < j; k++)
    {
        double newValue = score[i][k]+score[k+1][j];
        if (newValue > d) d = newValue;
    }
    double resultMax = Math.max(Math.max(a,b),Math.max(c,d));
    return resultMax;
}

public void fill()
{
    fillInit();
    fillRecursion();
}

private void fillInit()
{
    for (int i = 1; i < lengthSeq; i++)
    {
        score[i][i-1] = 0;
    }
    for (int i = 0; i < lengthSeq; i++)
    {
        score[i][i] = 0;
    }
}

private void fillRecursion()
{
    for (int j = 1; j < lengthSeq; j++)
    {
        int i = 0;
        for (int jDiagWalker = j; jDiagWalker < lengthSeq;
            jDiagWalker++)
        {
            score[i][jDiagWalker] = max(i,jDiagWalker);
            i++;
        }
    }

    printScore();
}
}

```

```

public void traceback()
{
    tracebackInit();
    tracebackRecursion();
}

private void tracebackInit()
{
    stack = new Stack();
    stack.push(new StackElement(0,lengthSeq-1));
}

private void tracebackRecursion()
{
    StackElement e = null;
    while(!stack.empty())
    {
        e = (StackElement)stack.pop();
        if (e.getI() >= e.getJ()) continue;
        else if (score[e.getI()+1][e.getJ()] ==
                score[e.getI()][e.getJ()])
            stack.push(new StackElement(e.getI()+1, e.getJ()));
        else if (score[e.getI()][e.getJ()-1] ==
                score[e.getI()][e.getJ()])
            stack.push(new StackElement(e.getI(), e.getJ()-1));
        else if (score[e.getI()+1][e.getJ()-1]
                + isBasePair(e.getI(),e.getJ())
                == score[e.getI()][e.getJ()])
        {
            basePairStack.push(new StackElement(e.getI(),
                e.getJ()));
            stack.push(new StackElement(e.getI()+1, e.getJ()-
                1));
        }
        else
        {
            for (int k = e.getI()+1; k < e.getJ(); k++)
            {
                if (score[e.getI()][k] + score[k+1][e.getJ()]
                    == score[e.getI()][e.getJ()])
                {
                    stack.push(new StackElement(k+1, e.getJ()));
                    stack.push(new StackElement(e.getI(), k));
                    break;
                }
            }
        }
    }
}

```

```

}

private void printScore()
{
    System.out.print("#\t");
    for (int j = 0; j < lengthSeq; j++)
        System.out.print(sequence.charAt(j)+"\t");
        System.out.println("\n");
    for (int i = 0; i < lengthSeq; i++)
    {
        System.out.print(sequence.charAt(i)+"\t");
        for (int j = 0; j < i-1; j++) System.out.print("#\t");
        for (int j = i-1; j < lengthSeq; j++)
        {
            if (j >= 0) System.out.print(score[i][j)+"\t");
        }
        System.out.println("\n");
    }
    System.out.println("Next State\n");
    BufferedReader din = new BufferedReader(new
        InputStreamReader(System.in));
}

private void paintBasePairStack()
{
    while (!basePairStack.empty())
    {
        StackElement e = (StackElement)basePairStack.pop();
        System.out.println(sequence.charAt(e.getI()+
            "+e.getI()"+ "-" + sequence.charAt(e.getJ()+
            "+e.getJ()");
    }
}

public static void main(String args[])
{
    //TYPE SEQUENCE HERE!
    Nussinov algorithm = new
        Nussinov("AAACAUGAGGAUUACCCAUGU");
    algorithm.fill();
    algorithm.traceback();
    algorithm.paintBasePairStack();
}
}

```

```
//Class Object representing a pair (i,j) in a stack
class StackElement
{
    public int I;
    public int J;

    public StackElement(int x,int y)
    {
        this.I = x;
        this.J = y;
    }

    public int getI()
    {
        return I;
    }
    public int getJ()
    {
        return J;
    }
}
//End of StackElement
```

## APPENDIX D – GARNAFOLD SOURCE CODE

### GARNAFold Source code

Important sections of GARNAFold implementation :

```
/*
 * This file is part of GARNAFold.
 * Main.java
 * Created on 01 June 2005, 13:54
 */
package GARNAlgorithm;
/**
 * @author Asim Khan
 * @course MSc Computer Science
 */
import java.io.*;
import java.util.Stack;
import java.util.*;
import java.lang.Object.*;

//importing some supporting classes from JGAP
import GARNAFold.ga.FitnessFunction;
import GARNAFold.ga.Configuration;
import GARNAFold.ga.InvalidConfigurationException;
import GARNAFold.ga.Gene;
import GARNAFold.ga.Genotype;
import GARNAFold.ga.Chromosome;
import GARNAFold.ga.impl.DefaultConfiguration;
import GARNAFold.ga.impl.IntegerGene;

public class Main
{
    public String sequence = "";
    private int lengthSeq = 0;

    // putting a constraint on the number of base pairs
    private int maxBasePairs = 0;

    //defining variables to keep a count of the individual
    nucleotide statistics
    private int noOfAs = 0;
    private int noOfUs = 0;
    private int noOfCs = 0;
    private int noOfGs = 0;
```

```

//defining variables keeping track of the number of
//possible pairs
private int possAU = 0;
private int possCG = 0;

//defining variables keeping track of unpaired
//nucleotides statistics
private int possUnpairedAs = 0;
private int possUnpairedUs = 0;
private int possUnpairedCs = 0;
private int possUnpairedGs = 0;

//matrix to save the chromScore. It is used to define
//pairs.
private double[][] chromScore = null;

//private Stack stack = null;
private Stack basePairStack = new Stack();
Gene[] genes;

// for stepping through the algorithm
protected String state = "";

//constraint
public static int maxStemSize = 0;

// Creates a new instance of Main
public Main(String sequence)
{
    this.sequence = sequence.toUpperCase();
    this.lengthSeq = sequence.length();
    this.initNCount();
    this.initChromScore();
}

//main method
public static void main(String[] args) throws
    InvalidConfigurationException
{
    //Enter Sequence here !
    Main algorithm = new Main("CUACUCUGGAGAGUGACAAU
        CAUGUAUAACUAAAUUUGAUUGACACUUCUGUGAGUAG");

    int maxEvalAllowed = 300;
    maxStemSize = 5;
    algorithm.printSequenceStats();//printing sequence
//details
    algorithm.gAlg(maxEvalAllowed);//runs GA
    algorithm.printChrom();// print chromosome
}

```

```

        algorithm.paintBasePairStack();// print base pair
                                         stack
    }

//method to print sequence statistics
private void printSequenceStats()
{
    System.out.println("length of sequence:
                                         "+lengthSeq);

    System.out.println("As: "+noOfAs);
    System.out.println("Us: "+noOfUs);
    System.out.println("Cs: "+noOfCs);
    System.out.println("Gs: "+noOfGs);

    possAU = Math.min(noOfAs,noOfUs);
    possCG = Math.min(noOfCs,noOfGs);

    System.out.println("Possible A-U pairs: "+possAU);
    System.out.println("Possible C-G pairs: "+possCG);

    if (noOfAs >noOfUs)
    {
        possUnpairedAs = Math.abs(noOfAs-possAU);
    }
    else possUnpairedUs = Math.abs(noOfUs-possAU);

    if (noOfCs >noOfGs)
    {
        possUnpairedCs = Math.abs(noOfCs-possCG);
    }
    else possUnpairedGs = Math.abs(noOfGs-possCG);

    System.out.println("Free As: "+possUnpairedAs);
    System.out.println("Free Us: "+possUnpairedUs);
    System.out.println("Free Cs: "+possUnpairedCs);
    System.out.println("Free Gs: "+possUnpairedGs);
}

//count the instances of As, Us, Cs and Gs

private void initNCount()
{
    for (int ct =0; ct<lengthSeq;ct++)
    {
        if (sequence.charAt(ct)=='A')
        {
            this.noOfAs++;
        }
        if (sequence.charAt(ct)=='U')

```

```

        {
            this.noOfUs++;
        }
        if (sequence.charAt(ct)=='C')
        {
            this.noOfCs++;
        }
        if (sequence.charAt(ct)=='G')
        {
            this.noOfGs++;
        }
    }
}

private Chromosome getStChrom()
{
    //creating genes
    Gene[] genes = new Gene[lengthSeq];
    //initialising all Genes randomly
    for (int i=0;i<genes.length;i++)
    {
        genes[i] = new IntegerGene(0, lengthSeq+1);
    }

    Chromosome chrom = new Chromosome(genes);
    System.out.println("StringChrom: "+chrom);
    int a= 0;
    int b = 0;
    for (int j=0;j<genes.length;j++)
    {
        a = (Integer.valueOf(chrom.getGene(j)
            .getAllele().toString())).intValue();
        b = (Integer.valueOf(chrom.getGene(j+1)
            .getAllele().toString())).intValue();
        if (a<b)
        {
            getStChrom();
            break;
        }
        else break;
    }
    System.out.println("StringChrom: "+chrom);
    return chrom;
}

//Setting up the configuration of the GA
private void gAlg(int maxEvalAllowed) throws
    InvalidConfigurationException
{

```

```

Configuration conf = new DefaultConfiguration();

//Setting Fitness Function
FitnessFunction myFunc = new
                                RNAFitness(this.sequence);
conf.setFitnessFunction(myFunc);

//creating genes
Gene[] genes = new Gene[lengthSeq];

//initialising all Genes randomly
for (int i=0;i<genes.length;i++)
{
    genes[i] = new IntegerGene(0, lengthSeq+1);
}

//setting up chromosome
Chromosome chrom = new Chromosome(genes);

conf.setSampleChromosome(chrom);

//setting up population
conf.setPopulationSize(100);

Genotype population =
    Genotype.randomInitialGenotype(conf);

//Printing population at every evolution
for (int i=0;i<50;i++)
{
    System.out.println("population:
        "+population.getPopulation().getChromosome(i));
}

//Printing population configuration
System.out.println(population.getConfiguration());
int ctr = 0;
double previous = 0;
for(int i = 0; i< maxEvalAllowed;i++)
{
    population.evolve();
    //Printing evolution with its best chromosome
    System.out.println(i+": Best way so far " +
        population.getFittestChromosome());
    System.out.println(1/conf.getFitnessFunction()
        .getFitnessValue(population
        .getFittestChromosome()));
}

```

```

//Printing the normalised fitness values
System.out.println(Integer.MAX_VALUE / 2 -
                    conf.getFitnessFunction().getFitnessValue
                    (population.getFittestChromosome()));

if(conf.getFitnessFunction().getFitnessValue
   (population.getFittestChromosome()) == previous)
{
    ctr++;
}
else
{
    ctr = 0;
}
if (ctr == 50 || previous==0)
{
    population.evolve(20);
    ctr = 0;
}

previous = conf.getFitnessFunction()
            .getFitnessValue(population
                            .getFittestChromosome());
}
Chromosome bestSolutionSoFar = population
                            .getFittestChromosome();

System.out.println("Optimal way " +
                    bestSolutionSoFar);
//Printing chromosome and fitness stats of best
                    solution
System.out.println("Score " +
                    (Integer.MAX_VALUE / 2 -
                    conf.getFitnessFunction()
                    .getFitnessValue(bestSolutionSoFar)));
this.printStructure(population.
                    getFittestChromosome());

}

//Method isBasePair return 1 if true, else returns 0
private int isBasePair(int i, int j)
{
    char letterI = sequence.charAt(i);
    char letterJ = sequence.charAt(j);

```

```

        if (letterI == 'A' && letterJ == 'U' ||
            letterI == 'U' && letterJ == 'A' ||
            letterI == 'C' && letterJ == 'G' ||
            letterI == 'G' && letterJ == 'C') return 1;
        else return 0;
    }

//Method initialises the chromScore matrix to -1
private void initChromScore()
{
    //
    this.chromScore = new
                        double[lengthSeq][lengthSeq];

    // just for vision
    for (int i = 0; i < lengthSeq; i++)
    {
        for (int j = 0; j < lengthSeq; j++)
        {
            chromScore[i][j] = 0;
        }
    }
}

private void fillInit()
{
    for (int i = 1; i < lengthSeq; i++)
    {
        chromScore[i][i-1] = 0;
    }
    for (int i = 0; i < lengthSeq; i++)
    {
        chromScore[i][i] = 0;
    }
}

//method to print the predicted structure
private void printStructure(Chromosome chrom)
{
    int[] temp = new int[lengthSeq];
    for(int i=0; i<lengthSeq;i++)
    {
        temp[i] = Integer.valueOf(chrom.getGene(i)
                                .getAllele().toString()).intValue();
    }
    for (int j=0;j<lengthSeq;j++)

```

```

    {
        if (temp[j] >=lengthSeq)
        {
            if(j<lengthSeq-1 && temp[j+1]<lengthSeq)
            {
                System.out.print(sequence.charAt(j));
                System.out.println(j+" ");
            }
            else
            {
                System.out.print(sequence.charAt(j));
                System.out.print(j+" ");
            }
        }
        else
        {
            if (j<temp[j])
            {
                System.out.println("
                "+sequence.charAt(j)+j+"-"+sequence
                .charAt(temp[j])+temp[j]);
            }
        }
    }
}

```

```

//method to print chromosome
private void printChrom()
{
    System.out.print("#\t");
    for (int j = 0; j < lengthSeq; j++)
        System.out.print(sequence.charAt(j)+"\t");
    System.out.println("\n");
    for (int i = 0; i < lengthSeq; i++)
    {
        System.out.print(sequence.charAt(i)+"\t");
        for (int j = 0; j < i-1; j++)
            System.out.print("#\t");
        for (int j = i-1; j < lengthSeq; j++)
        {
            if (j >= 0) System.out.print(chromScore[i][j]+
            "\t");
        }
        System.out.println("\n");
    }
    System.out.println("Next State.....\n");
    BufferedReader din = new BufferedReader(new
        InputStreamReader(System.in));
}

```

```

        try
        {
            din.readLine();
        }
        catch(IOException e){};
    }
}

//creating chromosome structure
class ChromStructure
{
    public char i;
    public char j;
    public char unpaired;

    public ChromStructure(char x,char y)
    {
        this.i = x;
        this.j = y;
    }
    public ChromStructure(char x)
    {
        this.unpaired=x;
    }

    public char getI()
    {
        return i;
    }
    public char getJ()
    {
        return j;
    }

    public char getUnpaired()
    {
        return unpaired;
    }
}
//End of ChromStructure

//Creating Gene Structure
class GeneStructure
{
    public int i;
    public int j;

    public GeneStructure(int x,int y)
    {

```

```

        this.i = x;
        this.j = y;
    }

    public int getI()
    {
        return i;
    }
    public int getJ()
    {
        return j;
    }
    public void setI(int input)
    {
        i = input;
    }
    public void setJ(int input)
    {
        j = input;
    }
}
//End of GeneStructure

//The important class for fitness claculation
class RNAFitness extends FitnessFunction
{
    public String seq = ""; //sequence
    int lengthSeq;//length of sequence
    public double[][] chromScore; //n-by-n matrix for the
                                                sequence

    HashMap possPairs = new HashMap();//to save all
                                                possible pairs
    Stack stack = new Stack();//to save accepted pairs

    // Creates a new instance of RNAFitnessFunction
    public RNAFitness(String sequence)
    {
        this.seq = sequence;
        this.lengthSeq = seq.length();
    }

    //method check is two nucleotides pair up
    public int isBasePair( int i, int j)
    {
        char letterI = seq.charAt(i);
        char letterJ = seq.charAt(j);

        if (letterI == 'A' && letterJ == 'U' || letterI ==

```

```

        'U' && letterJ == 'A' ||
letterI == 'C' && letterJ == 'G' || letterI == 'G'
        && letterJ == 'C') return 1;
else return 0;
}

//the main fitness evaluation method
public double evaluate(Chromosome chrom)
{
    double fitness = 0;
    chromScore = new double[lengthSeq][lengthSeq];
    //double[][] fitGene = new
        double[lengthSeq+1][lengthSeq+1];
    int[] val = new int[chrom.size()];

    Vector temp = new Vector();

    //minimum 5 gene restriction
    if(chrom.size()<4)
    {
        throw new IllegalArgumentException("Chromosome
            for RNA pairig must have more than 4 genes");
    }

    //Creating a matrix fitGene from the chromosome for
        fitness calculation

    //initialise matrix
    for(int i=0;i<lengthSeq;i++)
    {
        for(int j=0;j<lengthSeq;j++)
        {
            chromScore[i][j]= 0;
        }
    }

    //converting chromosome into a matrix
    for(int x=0;x<lengthSeq;x++)
    {
        int geneInt = Integer.valueOf(chrom.getGene(x)
            .getAllele().toString()).intValue();
        if(geneInt < lengthSeq)
        {
            chromScore[x][geneInt]=1;

            //saving the chromosome presumed pairs in
                the possPairs
            possPairs.put(String.valueOf(x),
                chrom.getGene(x).getAllele().toString());
        }
    }
}

```

```

        //printing pairs
        System.out.println("(" +String.valueOf(x)+",
            "+possPairs.get(String.valueOf(x))+")");

        if(isBasePair(x, geneInt)==1)
        {
            stack.push(String.valueOf(x)+", "
                +possPairs.get(String.valueOf(x)));
        }

    }
    else
    {
        possPairs.put(String.valueOf(x),
            String.valueOf(lengthSeq+1));
    }
}

//Applying hard constraints
fitness = this.applyHardConstraints(chromScore,
                                    chrom); //Testing for now

//clearing up possairs and stack
possPairs.clear();
stack.clear();

//resetting negative values to a small positive
value as the fitness function takes only non
negative values

if (fitness<0)
{
    fitness=0.1;
}

return fitness;
}

//method calculaes the hard constraints
private double applyHardConstraints(double[][]
                                    chromScore, Chromosome chrom)
{
    double fitness = 0;
    double [] fitGene = new double[lengthSeq];

    //initialising fitGene
    for (int i=0; i<lengthSeq;i++)
    {
        fitGene[i]=0;
    }
}

```

```

}

//Checking constraint 1 :- Watson-Crick pairing
//Constraint 2:(i,i) not possible
//Constraint 3: No sharp loops (minimum gap = 4
                                neucleotides)
//Constraint 4: No overlapping pairs i.e.
    //If P contains (i, j), then it cannot contain
                                (i, k)
    //if k is not equal to j or (k, j) if k is not
                                equal to i
//Constraint 5: No Pseudoknots
//Constraint 6: Checking stack pairs - encouraging
                                stems
//Constraint 7: Checking the order of pairs for
                                consistency

for (int i =0; i<lengthSeq;i++)
{
    int getKey = i;
    int getVal = (Integer.valueOf(chrom.getGene
        (getKey).getAllele().toString()).intValue());
    if(getVal<lengthSeq) //Values greater than
        length refer to no paired nucleotides
    {
        int getValofVal = (Integer.valueOf(chrom
            .getGene(getVal).getAllele()
                .toString()).intValue());
        if (isBasePair(getKey, getVal)==1 //Constraint1
            && getKey!=getVal //Constraint 2
            && getKey == getValofVal
                // (representation issue)
            && Math.abs(getKey-getVal)>=4)
            // Constraint 3
        {
            //boosting fitness value of chromosomes
            mating these constraints
            fitGene[getKey]=fitGene[getKey]
                +(lengthSeq)*50000;

            //Constraint 4: No Overlapping pairs
            boolean cons4 = true;
            int ctr=0;
            for(int j=0;j<lengthSeq;j++)
            {
                //if value of getKey is j && j!=
                getVal
                if((Integer.valueOf(chrom.

```

```

        getGene(j).getAllele().toString()
            .intValue())== getVal
        && j!=i
        && (Integer.valueOf(chrom
            .getGene(j).getAllele()
                .toString()).intValue()
            <lengthSeq)
    {
        //fitGene[j] = fitGene[j]-
            lengthSeq;//100;
        ctr++;
    }
}
if (ctr>0)
{
    fitGene[i]=fitGene[i]-lengthSeq;
    cons4 = false;
}
else
{
    fitGene[i]=fitGene[i]+
        lengthSeq*1000;
}

//Applying constraint 5: No pseudoknot
int ctrl = 0;
boolean cons5 = true;
for(int j=getKey+1;j<getVal;j++)
{
    //for (i,j) checking knots of (a,b)
        where a<i<b<j
    for(int k=getKey-1;k>=0;k--)
    {
        if((Integer.valueOf(chrom.
            getGene(k).getAllele()
                .toString()).intValue())==j)
        {
            fitGene[i]=fitGene[i]-
                lengthSeq;
            ctrl++;
        }
        else fitGene[i]=fitGene[i]+
            lengthSeq;
    }

    //for (i,j) checking knots of (a,b)
        where i<a<j<b
    for(int m=getVal+1;m<lengthSeq;m++)
    {

```

```

        if((Integer.valueOf(chrom
            .getGene(m).getAllele()
            .toString()).intValue())==j)
        {
            fitGene[i]=fitGene[i]-
                lengthSeq;
            ctrl1++;
        }
    }
}
if (ctrl1>0)
{
    fitGene[i]=fitGene[i]-
        lengthSeq*1000;
    cons5 = false;
}
else
{
    fitGene[i]=fitGene[i]+
        lengthSeq*100;
}

//Constraint 6

if(i<lengthSeq-1 && getVal>0 &&
    getVal<lengthSeq
    && cons4 == true
    && cons5 == true)
{
    if ((Integer.valueOf(chrom
        .getGene(i+1).getAllele()
        .toString()).intValue())==getVal-1
        && i+1 ==
        (Integer.valueOf(chrom
            .getGene(getVal-1).getAllele()
            .toString()).intValue())
        && getVal-1 != i+1)
    {
        fitGene[i] =
            fitGene[i]+lengthSeq;//43;
    }
}
if(i>0 && getVal<lengthSeq-1
    && cons4 == true
    && cons5 == true)
{
    if ((Integer.valueOf(chrom
        .getGene(i-1).getAllele()
        .toString()).intValue())

```

```

                                                    ==getVal+1
        && i-1 ==
            (Integer.valueOf(chrom
                .getGene(getVal+1).getAllele()
                .toString()).intValue())
        && getVal+1 != i-1)
    {
        fitGene[i] =
            fitGene[i]+lengthSeq;
    }
}

//Constraint 7
int ctr2=0;
for (int j=i+1;j<lengthSeq;j++)
{
    if(getVal<lengthSeq
        && (Integer.valueOf(chrom
            .getGene(j).getAllele()
            .toString()).intValue())
            <lengthSeq
        && (Integer.valueOf(chrom
            .getGene(j).getAllele()
            .toString()).intValue())
            > getVal)
    {
        fitGene[i] = fitGene[i]-
            lengthSeq*5000;//fitGene[j]-100;
        ctr2++;
    }
    else
    {
        fitGene[j]=fitGene[j]+
            lengthSeq;
        //representation constraint
        if (getKey == getValofVal)
        {
            fitGene[j]=fitGene[j]+
                lengthSeq;
        }
    }
}
if (ctr2>0)
{
    fitGene[i]=fitGene[i]-
        lengthSeq*500;
}
else
{

```

```

        fitGene[i]=fitGene[i]+
                                lengthSeq*5000;
    }
}

else//if not base pair etc
{
    fitGene[getKey]=fitGene[getKey]-
                                lengthSeq*500;
}
}
}

//Adding presumed AFM constraint of stem length
int temp = lengthSeq;
int prev = lengthSeq;
int ctr=0;
int[] ch = new int[lengthSeq];
for (int i=0;i<lengthSeq;i++)
{
    temp = (Integer.valueOf(chrom.getGene(i)
                            .getAllele().toString()).intValue());
    ch[i] = temp;
}

//Now we have an array ch with the partner
                                nucleotides at all indexes
for (int i=0;i<lengthSeq;i++)
{
    if((ch[i]<lengthSeq && i>0 && ch[i-
                                1]>=lengthSeq)
        ||
        (ch[i]<lengthSeq && i==0))
    {
        ctr =1;
        for(int j=i+1;j<lengthSeq;j++)
        {
            if((ch[j-1]-1) == ch[j])
            {
                ctr++;
            }
            else j=lengthSeq;
        }
        if(ctr>maxStemSize)//applying constraint
        {
            // excluding unwated stems from the
                                search space by
            // declaring it as nt fit
            fitGene[i]=fitGene[i]-10000;
        }
    }
}

```

```
        }
    }
}

//calculating total fitness of the chromosome
for (int i=0;i<lengthSeq;i++)
{
    fitness = fitness + fitGene[i];
}

return fitness; //fitness of the chromosome
}
}
//End of RNAFitness Class
```