DESIGN AND ANALYSIS OF BIOLOGICAL SEQUENCES USING CONSTRAINT HANDLING RULES

by

Maryam Bavarian

B.Sc., Sharif University of Technology, 2003

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Spring 2006

APPROVAL

Name:	Maryam Bavarian
Degree:	Master of Science
Title of thesis:	Design and analysis of biological sequences using Constraint Handling Rules

Examining Committee: Dr. Greg Mori
Chair

Dr. Veronica Dahl, Senior Supervisor

Dr. Fred Popowich, Supervisor

Dr. Diana Cukierman, SFU Examiner

Date Approved: Apr. 05/06

Abstract

The need for processing biological information is rapidly growing, owing to the masses of new information in digital form being produced at this time. Old methodologies for processing it can no longer keep up with this rate of growth. We present a novel methodology for solving an important bioinformatics problem, which has been proved to be computationally hard: that of finding a RNA sequence which folds into a given structure. Previous solutions to this problem divide the whole structure into smaller substructures and apply some techniques to resolve it for smaller parts, which causes them to be slow while working with longer RNAs. We prove that by using a set of simple CHR rules we are able to solve this problem and obtain an approximate but still useful solution more efficiently. We expect the results we present to be applicable, among other things, to in vitro genetics and to drug design.

keywords: Constraint Handling Rules, RNA secondary structure, RNA secondary structure design.

Acknowledgments

I want to express my gratitude to Dr. Veronica Dahl who has been more than a supervisor to me. She has always supported and encouraged me through my studies. Having the chance to know her and work with her was one of the best things that has happened to me.

I owe most of my achievements and success to my wonderful parents who have always been there for me. Their love does not have any borders and I am blessed to have such great parents.

Last but not least I want to thank my dear sister Sara and my beloved fiancé Payam whose love and support enabled me to complete this work.

Contents

List of Tables

List of Figures

List of Programs

Chapter 1

Introduction

What makes artificial intelligence so distinguished is its capacity to perform tasks that are normally viewed as requiring intelligence, and also to allow in many cases for declarative specification of a problem to become executable, largely invisibly to the user [2]. Semantic use of symbols other than numerical and character variables provide the means for coding highly structured applications. Examples of programming languages for coding artificial language programs are Prolog and Lisp which enable us to implement self-modifying and recursive programs [52]. Here our focus is on Prolog and Constraint Handling Rules (CHR) where the latter is a bottom-up framework using Prolog engine that gives us even more capability and ease in artificial intelligence programming. Using Prolog in particular gives us the benefit that the information in qualitative and quantitative statements need not to be encoded as simple structures such as strings, arrays, etc, but can be stated in high level ways which make sense to humans.

Searls in [53] has proposed four broad roles for tools and techniques of language theory in examining the functional aspects of the language of biological sequences, which are

- *Specification*: use of formalisms such as grammars to indicate the nature and relative locations of features in a sequence mathematically and computationally.
- *Recognition*: use of grammars as input to parsers which are used for pattern-matching search.
- *Theory formation*: elaboration of domain theories to model biological structures and processes. Machine learning techniques could be expected to be the most useful with

regards to this role.

- *Abstraction*: to present a mathematical view of the language of biological sequences, as sets of strings that are meaningful in a biological system.

1.1 Problem Definition

RNA (Ribonucleic Acid) is a chemical similar to DNA (Deoxyribonucleic Acid) with several functions. RNA secondary structure is one of the three kinds of structural information of RNA that defines which nucleotides (the structural units of RNA and DNA) bind to each other. Through this thesis, we have mainly contributed in solving the problem of *RNA secondary structure design* which is defined as follows: given an RNA secondary structure, find an RNA sequence which folds onto the given structure. This problem is the inverse problem of RNA secondary structure prediction where given an RNA sequence, the corresponding secondary structure is asked for.

Complexity-wise this problem is believed to be computationally hard [5] but it has not been proven yet. The problem of RNA secondary structure design can be considered as a constraint satisfaction problem where constraint variables are the positions in RNA and the constraints include the base-pairs inside the RNA.

In this thesis we present a directly-executable system that is able to accept a valid structure of RNA and design a sequence for the given structure. We argue that within Searl's classification our system falls within first and third category: it falls within specification because we use grammars for describing the known nature of secondary structure of RNA, and also within theory formation, insofar as it induces possible structures for the RNA specified. Using constraint handling rules we specify a grammar to parse the secondary structure of RNA and design the RNA sequence accordingly.

1.2 Motivations

Before the discovery of catalytic RNA by Cech and Altman, RNA was considered as a passive molecule serving as an intermediate between DNA and proteins with some limited functions such as carrying information or forming structures. But their discovery opened new doors and completely changed scientist's view of RNA and now it is clear that RNA can have several functions such as acting as a catalyst in RNA splicing and cleavage, DNA cleavage, and controlling gene expression. According to the above, one can conclude that RNA has all the significant properties of proteins [34].

As the structure of RNA has significant effects on its functions, designing RNA molecules with specific structural properties is promising with many potential applications [5]. Nowadays, scientists are capable of replicating any RNA in a test tube [16] which would eventually help in finding new paths for drug design or have industrial use [5].

An RNA molecule that catalyzes a reaction (which mostly pertains to the cleavage of an RNA or DNA) is called a *ribozyme*. The fact that predicting RNA structure is much easier than proteins, makes ribozyme (catalytic RNA) engineering i.e., modifying the ribozymes in order to alter their activity, more straightforward than protein engineering. There are two major motivations for ribozyme engineering. First, ribozyme engineering can be used as a tool to understand the mechanisms of catalysis or principles of RNA folding. Second, it can facilitate the inhibition of gene expression [57].

Some of the other motivations include design of artificial RNA nanostructures, drug design [57, 5], and ribozyme therapy [34]. Moreover, finding a solution to the problem of RNA secondary structure design might assist in solving the same problem for DNA (due to their analogous structures) which consequently can be used in DNA self-assembly computation [61].

1.3 Our Contribution

Currently there are two methods for solving the problem of RNA secondary structure design: RNA-SSD [5], and RNAinverse [33]. As we will show in detail in Chapter 7, although these methods perform very well for shorter sequences (300-base limit for RNA-SSD and about 500-base limit for RNAinverse), since their complexity mostly depends on the complexity of the inverse problem which is RNA secondary structure prediction with the complexity of $O(n^3)$, they fail to generate results within their time limit. Here in our method which we have named chrRNA[1], we try to overcome this problem by using a set of CHR rules that are easily understandable by biologists, combined with a set of probabilities. We do not employ any of the algorithms for RNA secondary structure prediction in our method; instead, we use other heuristics by combining our rules with a set of probabilities.

[1]Later, we will show that we have actually implemented two systems, namely chr-statRNA and chr-loopRNA. To refer to both methods in general, we use the term chrRNA.

Our method also has the advantage that unlike RNA-SSD and RNAinverse it does not force the following restrictions:

- RNA-SSD designs the RNA by only using Watson-Crick base pairs.
- Both methods assume that there is no pseudoknot in the desired structure.

In chrRNA we assign probabilities to all possible base pairs, moreover by using constraints we allow the formation of simple pseudoknots inside the structure.

The format of the whole thesis is as follows: In Chapter 2, we briefly present biological background needed to understand and visualize this problem in biological context. Chapter 3 mainly discusses relatively new methods in logic programming and AI such as Constraint Handling Rules, Constraint Handling Rules Grammar, Assumption Grammars, and Concept Formation Rules and their possible application in biology. Here, our main focus is Constraint Handling Rules (CHR) and we explain its syntax and semantics in detail. In Chapter 4, the RNA secondary structure and its features are looked at in detail and we also discuss some of the state-of-the-art methods for predicting RNA secondary structure which is the counterpart of our problem of focus namely RNA secondary structure design. Following in Chapter 5 we introduce and compare the two current methods used for designing the secondary structure of RNA. Through Chapter 6 and Chapter 7, we show our own contributions to solve this problem within two approaches. Both approaches are stochastic and use CHR and Prolog rules. However while chr-statRNA utilizes the nucleotide compositions probabilities within a generic RNA, chr-loopRNA exploits the thermodynamic energies assigned to the loops to find the probabilities. Chapter 3 and Chapter 6 are based on the papers written by Bavarian and Dahl [11, 10]. Finally in Chapter 8, we present the results for both approaches and we compare them with the results by RNA-SSD and RNAinverse.

Chapter 2

Biological Background

The strong relationship between biology and medicine is what makes it not only fascinating but also extremely significant. Nowadays scientists have reached a point where with regular drugs, they are not able to provide the cure for the newly discovered and identified human diseases. It is known that the key to all these cures, lies in the science of genetics. Solving the current mysteries in biology would progress medicine enormously in a way that shall benefit all humankind. Some applications of biology are: diagnosis of disease by inspecting the DNA sequence and locating specific genes which are related to a certain disease or calculation of a disease risk in an individual; pharmacogenomics which is a fast-growing field in which different dosage and drugs are prescribed for different people according to their genome, i.e. the whole hereditary information of the organism encoded in DNA, which makes their responses to therapy dissimilar; identification of drug targets which are proteins whose functions can be modified selectively and help to cure a disease; and last but not least identification of missing or defective genes and replacement or supplying of its products [23].

Some distinguishing aspects of biology according to Cohen are [23]:

- No rules are without exceptions. A famous example of this, was the false belief that every gene is only responsible for one protein.
- Every phenomenon has a nonlocal component. An RNA secondary structure may be adopted by multiple RNA sequences. We will elaborate on this problem in the next chapters. But basically this implies the similarity in one level can not be generalized to the others.

- Every problem is tied to the other problems. An example of this is the fact that a simple change in a DNA sequence might lead to a completely distinct protein with different functionality.

Interdisciplinary fields like biochemistry, biophysics, and bioinformatics are the projections of biology in other fields of science. The benefit of such collaboration has been proved to be bilateral. Not only chemistry has made major strides in biology and physical explanations for biological phenomena been extremely useful, but at the same time it has become a source of inspiration in the original sciences involved. Here, computer science has been no exception at all. Computer science can provide biologists the tools to gather and interpret the enormous amount of data they are producing every day. On the other hand, DNA computers and neural networks have been great examples of the effort of computer scientists to project what they see in the human body through computers [23]. Specifically, computer science can benefit biology by analyzing biological data, modeling, and simulation to help explain biological behavior, while atomic interactions can be used as a model to build a molecular data processor as well as a model for computational algorithms and finally, contributions from biology can be used to create computer programs that can be used to operate on biological data [52].

Computational biology is the predecessor of what is called bioinformatics today. The work of earlier computational biologists is now widely used in everyday tools for biologists, from software for aligning chunks of DNA to that which predicts the secondary structure of RNA. Some examples of their works include proofs of algorithmic correctness, complexity estimates,etc [23].

Although a lot of work has been done by computational biologists and in recent years by bioinformaticians, the need of biologists for faster, more optimized and newer algorithms requires more computer scientists to dedicate their efforts in this field which would eventually be most beneficiary to human beings.

As indicated above, today's bioinformaticians follow the work of earlier computational biologists, but there are major differences between these two sciences and their corresponding scientists. One major difference between these two is the widespread use of the World-Wide Web in bioinformatics making all the immense databases containing biological data and the extremely useful web applications, available to the researchers throughout the world. Some examples of these databases are National Resource for Biotechnology Information (NCBI)[1] providing an integrated approach to the use of gene and protein sequence information, Protein Data Bank (PDB)[2] that contains information regarding newly discovered proteins. As for the available web applications, some of the most widely used ones are mfold RNA Folding Server [3] which predicts the secondary structure of RNA, BLAST [4] and FASTA [5] where both of them are widely used to search sequence similarity against nucleotide and protein databases.

According to one of the most popular taxonomies, living things are divided into three domains: Bacteria, Archaea and Eukarya (Eukaryote). Bacteria and archaea are prokaryotes (their cells do not have nuclei). There are common characteristics shared by all cells no matter their originating organisms or their shape, size, and behavior. In 1865 with the discovery of genes by Gregor Mendel the story of genomics began. From a biochemical point of view, genes are specific sequences of bases that are subparts of DNAs. These encode the recipes for making different proteins and then, the proteins determine our physical traits such as hair and eye color. Genes are estimated to comprise only 2% of the human genome and the remaining 98% is basically regions for which the functions are currently unknown.

Through the next sections, we will present a brief background for the most significant constituents of the cells.

2.1 DNA

The discovery of DNA happened in 1953 but it was not until 1989 that James Watson and Francis Crick produced the first three-dimensional model of DNA which was based on the data gathered by Rosalind Franklin. DNA (Deoxyribonucleic Acid) is a nucleic acid that contains the genetic instructions specifying the biological development of all cellular forms of life (and many viruses) working like an information archive in each organism. DNA is often referred to as the molecule of heredity, as it is responsible for the genetic propagation of most inherited traits and it is replicated and transmitted to the offspring

[1]http://www.ncbi.nlm.nih.gov/
[2]http://www.umass.edu/microbio/rasmol/pdblite.htm
[3]http://www.bioinfo.rpi.edu/applications/mfold/old/rna/form1.cgi
[4]http://www.ncbi.nlm.nih.gov/Education/BLASTinfo/information3.html
[5]http://www.ebi.ac.uk/fasta33/

during reproduction. DNA can be looked at as a template for building other DNAs and proteins. DNA is encoded with four building blocks, called *nucleotides* or *bases*, which are: A (Adenine), C (Cytosine), G (Guanine) and T (Thymine). The DNA molecules are usually very long and linear. Even in microorganisms they are typically about 10^6 bases long. *Chromosomes* are the separate physical molecules which are arranged to form DNAs inside the cells. All the genetic information inside chromosomes is called the *genome*. When a cell is replicated, the entire genome inside the cell is copied into the new cells.

DNA can be read and replicated and it is believed that much of the role that DNA plays in forming proteins depends on this. These characteristics make the linguistics treat a gene as a 'word' while a genome, the total genetic material of species as 'text' in DNA code.

Structurally, DNA is organized as two complementary strands or helices with hydrogen bonds between them. Directions along each strand are named 3' end and 5' end and the correct direction of translation from DNA to proteins is always from 5' end to 3' end. The structure of DNA is uniform. The important features of DNA are its secondary structure and interactions with a solvent.

The stretches of DNA that code for proteins are called *exons*. But not all the information inside the DNA is expressed as proteins or RNA, some regions of the DNA sequence are devoted to control the mechanisms. In eukaryotes, exons inside a gene are separated from each other by some non-genetic material called *introns*, the function of which are currently unknown to biologists. The boundaries of exons and introns are called *splice sites*. Reliable prediction of splice sites is also an important task since it allows determination of the uninterrupted genes and consequently the corresponding amino acid sequence. The exons are typically multiples of three nucleotides and the reason is that each triplet of bases called a *codon* is translated into a certain amino acid [39]. In Table 2.1, the 20 known amino acids are shown.

2.2 RNA

RNA (Ribonucleic Acid) is a chemical found in cells which codes for amino acid sequences, serving as intermediate in the synthesis of protein. Each RNA molecule is made up of four different compounds called nucleotides or bases, each noted with one of the letters A (Adenine), C (Cytosine), G (Guanine) and U (Uracil). During the transformation process of

Table 2.1: The 20 amino acids inside proteins

G(Gly): glycine	A(Ala)A: alanine	P(Pro): proline	V(Val): valin
I(Ile): isoleucine	L(Leu): leucine	F(Phe): phenylalanine	M(Met): methionine
S(Ser): Serine	C(Cys): cysteine	T(Thr): threonine	N(Asn): asparagine
Q(Gln): glutamine	H(His): histidine	Y(Tyr): tyrosine	W(Trp): trytophan
D(Asp): aspartic acid	E(Glu): glutamic acid	K(Lys): lysine	R(Arg): arginine

proteins, synthesis of RNA molecules are directed by DNA sequences. Under normal physiological conditions, this strand or helix of nucleotides is folded onto itself by hydrogen-bond pairings of the nucleotide A with U and C with G which are called *Watson-Crick* or *canonical base pairs.* Pairing also might happen between G and U but this is not very frequent (except for tRNA) and GU pairs are often referred to as *wobble base pairs.* The difference between Watson-Crick base pairs and wobble base pairs is that the pairing between the first category are through two hydrogen bonds while for the latter it is only one hydrogen bond. The nucleotides are often referred to as bases and each pairing is called a *base pair.* Apart from canonical base pairs and wobble base pairs, other base pairs such as UC or GA are also feasible but are quite rare.

Three types of structural information for RNA are:

- **Primary structure:** the sequence of nucleotides inside RNA. These can be used for distinguishing RNA sequences from each other.
- **Secondary structure:** in general secondary structure for a biopolymer shows whether or not individual molecules are connected to each other and in case of RNA, secondary structure defines which nucleotides are forming complementary base pairs with each other and which ones remain unpaired.
- **Tertiary structure:** the actual positions of molecules in three-dimensional space is generally called the tertiary structure.

The secondary and tertiary structure of RNA determine the RNA interaction with other cell components. A number of structural motifs are found inside RNA secondary structure such as helix, hairpin loop, bulge loop, internal loop, etc (Figure 2.1).

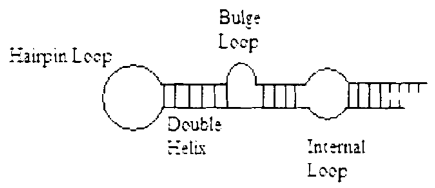


Figure 2.1: Common motifs in RNA secondary structure are: hairpin loop, bulge loop, internal loop, etc.

RNA mainly serves two biological purposes. These are information flow from DNA into proteins and acting as a structural component of ribosomes, an organelle in the cytoplasm of a living cell.

Important RNA families include:

- **Ribosomal RNA (rRNA)** combines with protein inside the cytoplasm and form a *ribosome* which serves as the site and carries the enzymes that help in the process of protein synthesis.
- **Transfer RNA (tRNA)** has the responsibility of reading the code and carrying the amino acid to the ribosome to be incorporated into the developing protein.
- **Messenger RNA (mRNA)** is used to carry the genetic code, the information on the primary sequence of amino acids, from the nucleus to the ribosomes.
- **Small nuclear RNA (snRNA)** refers to a number of small RNA molecules found in the nucleus and assists in several processes such as RNA splicing.

2.3 Proteins

Proteins are the molecules responsible for much of the structure and activities of organisms. From a chemical point of view, proteins are long polymers containing thousands of atoms. There are three major types of proteins. First, the structural proteins such as the ones that build the outer layer of human and animal skin. Second, proteins that function as catalyzers in chemical reactions such as enzymes, transport. And finally, proteins such as hemaglobin that work as storage. No matter what type of protein we are considering, the key to its functions is its three dimensional structure. The proteins are typically 200–400 amino acids long which means that they require at least 600–1200 letters for the DNA message to specify them (not including the introns). It is believed that the relationship between DNA and proteins is many-to-many.

The diverse structural and functional roles of proteins are due to the fact that unlike DNA and RNA they don't have a uniform structure. Taking a protein out of its normal condition e.g. by heating, unfolds the protein and makes it inactive. This is why our body needs a certain inner temperature to function properly [39].

The protein structure is also analyzed in three levels. Primary structure is the order of the protein's constituent amino acids. Secondary structure is the assignment of common structures like single spiral chains of amino acids called *alpha helices*, and *beta sheets* which are structures composed of two or more adjacent or parallel amino acids. Finally the tertiary structure is the exact conformation of the whole protein which is derived from the assembly and interactions of the helices and sheets. Some scientists have also added additional levels to this hierarchy such as supersecondary structures and domains.

It is the primary structure of the protein that wholly determines its secondary and tertiary structure. The function of proteins is mainly determined by its three-dimensional structure rather than the amino acid sequence. Therefore, in order to fully understand or predict the characteristics of any organism, we should first determine the three-dimensional structures of its proteins. The problem of protein folding is to define the relationship between primary and tertiary structure by establishing detailed rules. To this point, the process of protein formation can be easily described and here is nature's great leap from the linear world of genetic and protein sequences to the world of 3D proteins. This is one of the biggest mysteries of biology and solving this mystery would clearly have many rewarding implications such as designing exact drugs on a computer or more above transforming the practice of medicine to a more effective, personalized knowledge [15].

There are three known approaches for solving this problem. The first approach, namely the simulation approach, involves simulating the actual folding process with the mean force fields on all atoms and calculating the motion vectors for each atom (resulting from different forces such as hydrogen bonds, covalent bonds, etc). The problem with this approach is that it is highly time-consuming and as the actual folding of the proteins usually happens in the order of seconds or milliseconds, this needs a lot of sampling on hundreds of atoms. The second approach is protein structure prediction through energy minimization, in which one tries to find the fold with the minimum free energy and naturally defining a reliable energy function is a significant issue. According to some strong results, this problem is NP-hard [47], so for solving that, some simplified models have been used. Finally the third approach is called *protein threading* which aligns the protein with the ones for which the structure is currently known. This approach is a knowledge-based approach and it has been proved to be NP-complete [38, 4].

Although there is much to be done to solve the problem of protein folding and many of these efforts have not shown promising results, solving the same problem for the secondary structure could still give us some insight regarding the assembly of a whole protein in terms of its building blocks (alpha helices, beta sheets, etc). Fortunately the problem of protein secondary structure prediction is not as complex as tertiary structure prediction because of reductions in the number of potential conformations by several orders of magnitude, and the results might eventually lead to a solution for the original problem.

2.4 The Central Dogma

The central dogma of molecular biology considers how a sequence of DNA bases turns into a sequence of amino acids which in turn, forms proteins. The procedure is as follows: the information from a gene recipe is copied from a strand of DNA to a strand of messenger RNA (mRNA). The mRNA molecules then travel out of the nucleus and ribosome molecules read the genetic information inside mRNA and translate them into amino acids based on the genetic coding scheme (Table 2.1). This strand of amino acids is then folded to make the three dimensional structure of the protein.

The process of protein production resembles cascading machineries inside a factory where the output of each one is fed into the next as input. Here within the cells, there are three machineries for this whole process, *RNA polymerase, Splicesome, and Ribosome.* These machineries themselves are made of proteins and RNA. Inside Prokaryotes (bacterial cells) RNA is generated from DNA and proteins are the result of RNA translations. But in the more developed cells, Eukaryotes, there are some other additional steps in between [23].

$$DNA \Longrightarrow_{RNApolymerase} pre-mRNA \Longrightarrow_{Splicesome} RNA \Longrightarrow_{Ribosome} Protein$$

There are some important facts regarding these transformations. First of all, in the first transformation, genes are the material that would be processed by RNA polymerase machinery. The second fact is that splicesome can generate different RNA resulting in different proteins. And finally, some generated proteins of this process might work as a barrier in forming other proteins and on the other hand some of them might accelerate the production of others [23].

The major processes in protein production are [15]:

- **Replication** is the process of passing the information in the DNA molecule in one cell to the new cells, making sure that all the cells inside the body contain the same genetic code for making all the necessary proteins.
- **Transcription** happens when DNA sequences are transcribed by a biological machine called RNA polymerase into sequences called Pre-mRNA, in which the nucleotide T has changed to U (Uracil).
- **Splicing** is the process of transforming Pre-mRNA into mRNA by another biological machine called spliceosome.
- **Translation** is the process by which the genetic information of RNA, the codons, are read by ribosomes transferred into the proteins.
- **Folding** happens when amino acids twist and form the proteins.

In the next chapter, we will discuss several methods in logic programming we can use to analyze biological sequences.

Chapter 3

Constraint-Based Methods

The application to molecular biology of AI methods such as logic programming and constraint reasoning constitutes a fascinating interdisciplinary field which, despite being relatively new, has already proved quite fertile. For instance, the book Logic Grammars [2] was widely used to help discover the human genome [49]. The work on plant pathogen identification for Agriculture and Agri-Food Canada [62] yielded spectacular results: whereas with previous tools, the processing time increases exponentially with sequence length or number of sequences, here a novel algorithm is provided for which processing time increases linearly with the amount of data to be analysed. These methods can moreover be viewed as modules to be embedded within higher level ones, while still efficiently executable, descriptions of other interesting molecular biology problems (LifeIntel Explorer$^{\mathrm{TM}}$).

Over the past decade there has been a dramatic increase of collection rates for biological data, making the need for resorting to computational methods even more acute. Simultaneously, the intersection between logic programming and constraint reasoning has been maturing into extremely interesting methodologies, most notably Constraint Handling Rules, or CHR [30]. These methodologies have been applied to human language processing, through implementing Property Grammars [12, 14] (a linguistic formalism based on constraints between sentence constituents rather than on the traditional notion of phrase structure) in CHR, and through a parsing system for balanced parenthesis [13] and then to cognitive sciences, through generalizing these results into a general cognitive theory of concept formation [27] with applications to cancer diagnosis [8, 7], to medical report interpretation [58] and to concept extraction [24].

Our main contribution throughout this thesis involves applying CHR methodologies to the problem of genetic structure analysis, in particular, reconstructing a nucleotide sequence which can fold into a given RNA secondary structure: whereas previous approaches, while working well for short sequences, are computationally hard for longer ones (more than 500 nucleotides), our methodology by using heuristics obtains approximate but useful results in much faster time.

In this chapter, we share the expertise obtained in the course of this work in a pedagogical fashion. We first provide a short intuitive introduction to the main concepts involved in biological sequence analysis. Next we survey the CHR formalism itself, as well as its grammatical counterpart, CHRG [20], and CF formalism, exemplifying each with simplified subproblems within those we have addressed in the literature above referenced. We then discuss two major sequence processing problems (gene prediction, and protein structure) and compare the methods we advocate in this article with previously used methods.

3.1 Biological Sequence Analysis

We are interested in the general problem of protein generation from DNA. As discussed in previous chapter, DNA consists of a sequence of elements called nucleotides or bases, which are usually identified as the letters A (Adenine), C (Cytosine), G (guanine) and T (Thymine). In Section 2.1, and Section 2.4, we provided some background regarding DNA, its function, and the process, i.e. transcription, by which it is transformed into protein.

Computational methods have been used in each step of the transformation to simulate the work of the machineries (RNA polymerase, splicesome, and ribosome). Here we only focus on the language of DNA which consists of four words: A, C, T, and G. As mentioned before, nucleotide T would be transformed into nucleotide U in transcription step. The words of this language group together to code for different amino acids. For instance, the sequence UUU corresponds to the amino acid known as phenylalanine. These triplets of nucleotides which code for given amino acids are called codons. A first programming problem could be to implement the translation of codons into amino acids. This is an easy enough task for any computer scientist, and while challenging for biologists, and can be considered as a good introduction to interdisciplinary work. In a Prolog implementation, the core predicate contains table-like information such as:

```
translate([u,u,g],tryptophan).
translate([u,u,a],leucine).
translate([u,u,c],leucine).
```

We can also conceive a logic grammar version as well, containing such grammar rules as:

```
s([u,u,g]) --> [tryptophan].
s([u,u,a]) --> [leucine].
s([u,u,c]) --> [leucine].
```

As we can observe from this example, we are dealing with an ambiguous language: different codons can code for a given amino acid. Through the following sections, we present an overview of the proposed methodologies and specifically, we focus more on CHR methodology because of its key role in this thesis.

3.2 Assumption Grammars

Assumptions [26, 25] are similar to the Prolog primitive "assert" and "retract", the most notable exception being that, unlike "assert" and "retract", they are backtrackable. They can serve among other things to keep somewhat globally accessible information (an assertion can be used, or consumed, at any point during the continuation of the computation).

Assumption Grammars are basically like definite clause grammars, except that they can handle multiple accumulators invisibly, and that they possess linear and intuitionistic implications. They can also be described as logic grammars augmented with a) linear and intuitionistic implications scoped over the current continuation and b) hidden multiple accumulators useful in particular to make the input and output strings invisible. Intuitionistic assumption */1 temporarily adds a clause usable in later proofs. Such a clause can be used an indefinite number of times like asserted clauses except that it vanishes on backtracking. Linear assumption +/1 temporarily adds a clause usable at most once in later proofs. A built-in predicate -/1 was introduced to allow for the removal of either intuitionistic or linear assumptions.

Assumptions can be applicable in finding some patterns in biological sequences such as *tandem repeats*. Tandem repeat is a nucleotide sequence that results from a class of mutation event called tandem duplication which converts a stretch of DNA code (called the "pattern") into two or more copies, each following the preceding one in a contiguous fashion [53]. One of the reasons for identifying tandom repeats is that according to Biology and

Genetics sciences, tandem repeat occurs frequently in genomic sequences and it can have a potential role in gene regulation, including development of immune system cells. Many genetic diseases are also shown to be associated with uncontrolled expansions of tandem repeat patterns such as Huntington's disease and Friedreich's ataxia (FRDA). An example of a tandem repeat is the sequence AGCAGC.

Searls in [53] has introduced some grammar rules for finding tandem repeats. Searls' grammar rules can be represented in assumption grammar as following:

```
tandem_repeat -->[X],{push(X)}, tandem_repeat.
tandemrepeat --> repeat.
repeat --> {-stack([])},[].
repeat --> {pop(X)}, repeat, [X].
push(X):- -stack(Y), +stack([X|Y]).
pop(X):- -stack([X|Y]), +stack(Y).
```

Here we make use of a global variable as a stack, add on to it through assumption (noted '+'), and remove elements from it through consumption (noted '-'). Assumptions are available in some logic programming environments such as BinProlog and CHRGs.

Another example which is very similar to tandem repeat is *inverted repeat.* Inverted repeats are also common features of nucleic acids, which in the case of DNA result whenever a substring on one strand is also found nearby on the opposite strand [53]:

```
inverted_repeat --> [X],{push(X)}, inverted_repeat.
inverted_repeat --> repeat.
repeat --> {pop(X)},~[X],repeat.
repeat--> {-stack([])},[].
push(X):- -stack(Y),+stack([X|Y]).
pop(X):- -stack([X|Y]),+stack(Y).
```

Here, `~[X]` is defined as follows:

```
~[A] = [U], ~[U] = [A], ~[C] = [G], ~[G] = [C]
```

3.3 Constraint Handling Rule and Constraint Handling Rules Grammar

Constraint handling rules (CHR) is a concurrent committed-choice constraint logic programming language which has been proved to be useful for algorithms dealing with constraints [30]. By presenting a highly executable framework, CHR has tried to form a bridge between theory and practice in logic programming. It also provides programmers efficiently executable specifications by supporting rapid prototyping. To this day, CHR has been applied in several applications including theorem proving with constraints, combining forward and backward chaining, combining deduction and abduction, bottom-up evaluation with integrity constraint, etc.

3.3.1 Background

CHR was created in 1991 by Thom Frühwirth.The founders of CHR do not necessarily consider it as a new programing language but rather as an extension that blends in its host language such as the ones implemented in Prolog, Lisp, Haskell, or java [51]. The direct ancestors of CHR are Prolog, Constraint Programming, and concurrent committed choice logic programming. Term writing systems, Automated Theorem Proving, Chemical Abstract Machine, and production rule systems in general have had their influences in forming this language.

The factors that strengthen and to some extent make CHR a unique language are the combination of propagation and multi-head/multi-set transformation of logical formulae in a concurrent, guarded rule-based language.

Applications of CHR vary in a wide range from type systems and time tabling to ray tracing and cancer diagnosis [1]. Moreover by use of the logic terms, grammars can be described in human-like terms using CHR and are powerfully extended through (hidden) logical inference. Finally, CHR has been proved to be Turing-Complete which makes it possible to implement every algorithm in CHR with their best known time and space complexity [55].

CHR Grammars, or CHRG for short, is based on Constraint Handling Rules and was introduced in [19, 18] as a bottom-up counterpart to Definite Clause Grammars (DCGs) defined on top of CHR in exactly the same ways as DCGs take their semantics from and are implemented by a direct translation into Prolog. CHRG is executed as CHR programs that provide robust parsing with an inherent treatment of ambiguity.

3.3.2 Syntax

There are two sets of constraints (predicates of first-order logic) in CHR: *built-in constraints* and *CHR(user-defined) constraints*. The first set of constraints are solved by a given constraint solver while the latter are defined by CHR rules inside the program. The examples of the first set include =, *true*, and *false*. The general format of CHR rules is as follows:

```
Head ==> Guard | Body.
```

`Head` and `Body` are conjunctions of atoms (separated by commas) and `Guard` is a test constructed from (Prolog) built-in or system-defined predicates. The variables in `Guard` and `Body` occur also in `Head`. If the `Guard` is the constant "true", then it is omitted together with the vertical bar. Its logical meaning is the formula ($Guard \rightarrow (Head \rightarrow Body)$) and the meaning of a program is given by conjunction. There are three types of CHR rules:

- **Propagation rules** which add new constraints (body) to the constraint set while maintaining the constraints inside the constraint store for the reason of further simplification.
- **Simplification rules** which also add as new constraints those in the body, but remove as well the ones in the head of the rule.
- **Simpagation rules** which combine propagation and simplification traits, and allow us to select which of the constraints mentioned in the head of the rule should remain and which should be removed from the constraint set.

The factors strengthening CHR include the combination of propagation and multi-set transformation of logical formulae in a concurrent, guarded rule-based language. The rewrite symbols for the first two rules are respectively: `==>`, `<=>` and for simgation rules, the notation is `Head1\Head2<=>body`. Anything in `Head1` remains in the constraint set and anything in `Head2` is removed from the constraint set and `body` is added to the constraint store.

3.3.3 Semantics

We discuss two types of semantics for CHR programs:

- **Operational Semantics:** At run time, a CHR program starts execution with an initial state until either there are no more applicable rules or a contradiction occurs. A rule is considered applicable when all the constraints inside the head are matched by the ones inside the current goal and all the built-in predicates inside the guard are satisfied as well.

  At any given time during the execution, there might be more than one applicable rule. To determine which rule should be executed first, CHR acts in the same way as Prolog i.e. top-down in the textual order of the program. Another uncertainty arises while considering the order in which constraints of a query are processed. This problem is also easily solved by processing the constraints from left to right. When it is matched with one of the constraints inside the head of a rule, other constraints are examined and if all them are found inside the constraint store and the guard is evaluated *true*, the rule would be applied, adding new constraints to the constraint store.

- **Declarative Semantics:** It is possible to translate any CHR program into a first-order logic program formula using the declarative semantics of CHR. The logical reading (meaning) of the propagation rules was mentioned in Section 3.3.2, but for complicity it is also brought up here:

  1. Propagation rules:

  $$H \Rightarrow G|B \Longleftrightarrow \forall(G \rightarrow (H \rightarrow \exists\bar{y}B))$$

  2. Simplification rules:

  $$H \Leftrightarrow G|B \Longleftrightarrow \forall(G \rightarrow (H \leftrightarrow \exists\bar{y}B))$$

3.3.4 Sample Applications of CHR and CHRG

A string to be analyzed such as *"leucine tryptophan phenylalanine"* is entered as a sequence of constraints:

```
{token(0,1,leucine), token(1,2,tryptophan), token(2,3,phenylalanine)}
```

that comprise an initial constraint store. The integer arguments represent word boundaries, and a grammar for this intended language can be expressed in CHR as follows:

```
token(X0,X1,tryptophan)==> codon(X0,X1,[u,u,g]).
token(X0,X1,leucine)==> codon(X0,X1,[u,u,a]).
token(X0,X1,leucine)==> codon(X0,X1,[u,u,c]).
token(X0,X1,phenylalanine)==> codon(X0,X1,[u,u,u]).
```

We say that ambiguity is inherently treated because all possibilities will be expressed in the constraint store resulting from an ambiguous input. In the above example, for instance, both a codon [u,u,a] and a codon [u,u,c] will be found between points 0 and 1. The input and output arguments of the above translation example can be spread if using CHRG, which uses `::>` for the rewrite symbol:

```
token(tryptophan)::> codon([u,u,g]).
token(leucine)   ::> codon([u,u,a]).
token(leucine)   ::> codon([u,u,c]).
token(phenylalanine)::> codon([u,u,u]).
```

In Section 3.2, we introduced tandem repeats and their importance and showed an implementation of them using Assumption Grammars. Here using CHRG we present another implementation which unlike the other implementation which was only capable of determining whether a sequence is a tandem repeat or not, can identify a tandem repeat inside another sequence:

```
[X], string(Y) ::> string([X|Y]).
[X] ::> string([X]).

string(X),string(X) ::> tandem_repeat(X).
```

This implementation can also be extended to identify any number of tandem repeats:

```
[X], string(Y) ::> string([X|Y]).
[X] ::> string([X]).

tandem_repeat(X,C),string(X) <:> C1 is C+1 | tandem_repeat(X,C1).
string(X),string(X)::>tandem_repeat(X,2).
```

3.4 Concept Formation Grammars

In [27], a cognitive model of Concept Formation is introduced, which has been used for oral cancer diagnosis [9] and specialized into grammatical concept formation, with applications to property grammar parsing [24]. The grammatical counterpart of Concept Formation is an extension of CHRGs which dynamically handles properties between grammar constituents and their relaxation, as statically defined by the user. Let's first demonstrate this within the traditional framework of rewrite rules which implicitly define a parse tree. Whereas in CHRG we would write the following grammar:

```
[a] ::> determiner(singular).
[boy] ::> noun(singular).
[boys] ::> noun(plural).
[laughs] ::> verb(singular).

determiner(Number),noun(Number),v(Number) ::> sentence(Number).
```

To parse correct sentences such as "a boy laughs", in CFG, we can also allow incorrect sentences to be generated, but ask the system to point out to us which properties are violated by such incorrect sentences. This requires the following:

- A definition of all properties in terms of a system predicate `prop/2`.
- A declaration of what properties can be relaxed, done through the system's (Prolog) predicate `relax/1`, and for properties to be called through the system predicate `acceptable/1`.

For instance, an agreement property to check that the number of a subject's determiner (Ndet) coincides with that of the noun (Nn) and with that of the main verb (Nv), and which can be defined in Prolog as:

```
agreement(Ndet,Nn,Nv):- Ndet=Nn, Nn=Nv.
```

The property agreement must be expressed in terms of the system predicate `prop/2` as follows, with all concerned arguments grouped into a list:

```
prop(agreement,[Ndet,Nn,Nv]):- Ndet=Nn, Nn=Nv.
```

If we now want to relax this property, so that number mismatches are detected but do not block the parse, we can use the system predicate `relax/1`, as follows:

```
relax(agreement).
```

A property is then considered "acceptable" if it either succeeds, or if it fails but has been relaxed:

```
acceptable(prop(P,A)):-call(prop(P,A)); relax(P).
```

Therefore, properties must be tested through the system predicate `acceptable/1`. For our example, we would write:

```
determiner(Ndet), noun(Nn), v(Nv) ::>
acceptable(prop(agreement,[Ndet,Nn,Nv]))| sentence(Nv).
```

The system will now also accept sentences which do not agree in number, while pointing out the error in the list of violated properties, as a side effect of the parse. In the case of "a boys laughs", the agreement property will appear in the list of violated properties automatically constructed as a result of the parse.

Degrees of acceptability can also be defined using binary versions of "relax" and "acceptable", whose second argument evaluates to either true, false, or a degree of acceptability, according to whether (or how much of) the property is satisfied. This allows the user to relax specific constraints rather than types of constraints, by specifying right-hand side conditions on these binary counterparts.

Now that we have introduced this methodology, it is time to give some examples of its application in biological sequence analysis. According to Searls [54], in biology there are no rules without exceptions. For example, it was thought before that a gene can only be responsible for one protein, but now the recent work has shown that a gene may generate several proteins. According to these, using Concept Formation Rules features to the rules written for biological transformations might help in making these rules closer to what happens in real life. As an example we show how to use concept formation rules in finding *hairpin loops*, one of the common patterns seen in RNA structure. As shown in Figure 3.1, a hairpin loop consists of a stem which leads to a loop at the end. According to the biochemical laws the loop part should contain at least three nucleotides, but in some rare RNA

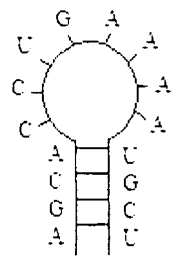


Figure 3.1: A hairpin loop

structures, it might happen that a loop contains only two nucleotides. A corresponding rule which also considers these rare cases is:

```
stem(X1,Y1,X2,Y2),loop(X,Y)==>
X is X2+1,Y is Y2-1, acceptable(prop(length,[X,Y])) | hairpin(X,Y).
```

While forming the secondary structure of a hypothetical RNA, one can easily relax this property by using `relax(length)`. In the following sections, you will see some of the large-scale applications of the methods proposed above in biology.

3.5 Gene prediction

The fundamental physical and functional unit of heredity is called a gene. It is an ordered sequence of nucleotides located in a particular position on a particular chromosome that encodes a specific functional product (i.e., a protein or RNA molecule). Genes determine many aspects of anatomy and physiology by controlling the production of proteins. Each individual has a unique sequence of genes, or genetic code. Discovery of genes that influence disease risk in human populations is of key importance to the pharmaceutical industry. This is because identifying the gene automatically identifies the protein in which alteration of function causes disease. For the pharmaceutical industry, this protein, together with other proteins that are part of the same physiological pathway, becomes a potential drug target. Once a drug target has been identified, modern techniques allow the pharmaceutical industry to rapidly screen large numbers of chemical compounds for action on this target. It is expected that drugs acting directly on the proteins, in which genetically determined alteration of function causes disease, will be highly effective in preventing or treating these diseases. This is one of the reasons why researchers in various branches of science are trying to identify genes. The problem of identifying genes responsible for certain diseases or characteristics has many difficulties. One of them is the fact that there are often several genes that are responsible for certain diseases. By studying the DNA of the individuals and their families having rare diseases one may be more successful in pinning down the main responsible genes [22].

The order in which the bases of DNA are linked in a gene is called the sequence of a gene. There exist two types of genes: RNA gene and protein coding gene which is our focus here. Recall from previous chapter that stretches of DNA that code for proteins are called *exons*. In eukaryotes, cells with membrane-bound nucleus, exons in a given gene are generally separated from each other by stretches of DNA that do not contain instructions for constructing proteins, they are called *introns*.

There are various approaches for the problem of gene finding but here our focus is on the computational methods. Computational gene finding is the process of identifying potential coding regions in an uncharacterized region of the genome. This area is still a subject of active research. There actually exist many different gene-finding software packages and no program is capable of finding everything. Common techniques include homology, combinatorial dynamic programming, probabilistic modeling especially Hidden Markov Models and neural nets but here another approach in logic programming is used.

Cohen has proposed a set of grammar rules for finding genes [22] and here we have translated the same grammar into CHRG in Program 3.1.

3.6 Protein structure

As seen in Section 2.3, protein architecture can be analyzed in three levels. Primary structure is the order of the amino acids in the sequence which has been formed by three codons translation. Secondary structure however represents how some of the amino acids in the sequence form common structures such as alpha helices, beta sheets, etc. Finally the tertiary structure is the exact conformation of a whole protein. Here we only discuss the problem of *protein secondary structure prediction*. What makes this problem rather difficult is that identical short sequences of amino acids can adopt different secondary structure in different contexts [52].

```
start_codon,gene_body, stop_codon ::> gene.
start_codon,stop_codon ::> gene.
[a,t,g] ::> start_codon.
[t,a,a] ::>stop_codon.
[t,a,g] ::>stop_codon.
[t,g,a] ::>stop_codon.
exon_body,left ::> gene_body.
exon_body ::> gene_body.
left ::> gene_body.
[a,g],intron_body,right ::> right.
[a,g],exon_body,left ::> right.
[a,g],left ::> right.
[a,g],right ::> right.
[a,g],exon_body ::> right.
[a,g] ::> right.
[g,t],intron_body,right ::> left.
[g,t],exon_body,left ::> left.
[g,t],exon_body ::> left.
[g,t],left ::> left.
[g,t],right ::> left.
[g,t] ::> left.
base ::> intron_body.
base ::> exon_body.
base,[a]::>base.
base,[c]::>base.
base,[g]::>base.
base,[t]::>base.
[a] ::> base.
[c]::> base.
[g]::> base.
[t]::> base.
```

Program 3.1: CHRG rules to identify a sample protein coding gene

```
res(X1),res(_),res(X2),res(X3),res(Y),res(X4),res(X5),res(_),res(X6) ::>
not_aromatic(X1),not_k(X1),
hydrophobic(X2),not_aromatic(X3),
not_p(X3),
not_aromatic(Y),not_p(Y),
not_p(X4), not_k(X4),
hydrophobic(X5),less_hydro(X1,X5),less_volume(X5,X3),
not_aromatic(X6),less_hydro(X6,X2) | alpha(Y) .
```

Program 3.2: A sample CHRG rule to form an alpha helix including 6 residues

Muggleton et al. have tried to solve this problem by using Inductive Logic Programming(ILP) [44, 46]. They have applied an ILP program to learn secondary structure prediction rules. The output of their program is a small set of rules which can predict which of the residues in the sequence are part of the alpha helices. These set of rules can also be translated into CHRG and can be used widely to discover the secondary structure of the requested protein. A sample rule in CHRG format is brought in Program 3.2. In this example, `res/1` represents a residue and the predicates in the guard (`not_aromatic/1`, `not_k/1`, `hydrophobic/1`, etc) are the chemical characteristics that each residue should have to form an alpha helix together. One specific advantage of using CHRG for this example is that here we can also make use of Concept Formation Rules to allow the formation of alpha helices when not all the conditions are satisfied which is probable in cases of biological data.

3.7 Discussion

We have covered several high-level methods for pattern description of biological sequences, and exemplified the advantages of these methods for several concrete such problems. We have shown how to translate codons to amino acids using DCGs, how Assumption Grammars allow us to modularize stack manipulation in global terms, which we used for implementing tandem and inverted repeats, how constraint handling rules promote direct bottom-up execution of the same problem, as well as having a grammatical version which we used for more direct amino acid string translations, and how Concept Formation Grammars can in particular accomodate rules with exceptions.

We consider the main advantage of these methods to be the coupling of direct executability with economy of expression within high level specifications that are meaningful for humans and thus can promote quick prototyping of specialized, interdisciplinary knowledge. On the other hand, the introduced methodologies has most of the time the disadvantage of inefficiency in run-time. For future work, some intermediate systems can be designed to automatically translate these high-level languages into low-level ones. This way we would have methods that are not only understandable and more user friendly but at the same time efficient and practical.

Chapter 4

RNA Secondary Structure

One of the significant benefits of understanding the secondary structure of RNA is to determine its chemical and biological properties. Although it is not yet possible to reliably predict RNA tertiary structure for reasons similar to the difficulties behind proteins tertiary structure prediction, there are fairly good RNA secondary structure prediction algorithms available. Clearly, the tertiary structure of RNA is much more useful and gives us a better insight of RNA functions, but while it is still quite difficult to predict, researchers use the secondary structure to explain most of the functionalities of RNA. There are a number of applications for RNA secondary structure determination [15]:

- Similarity in the secondary structure usually determines the similarity in functions. Therefore, it is reasonable to use RNA secondary structure for classifying their function.

- As ribosomal RNA (rRNA) are very ancient molecules and have a highly conserved secondary structure, it is possible to align the ribosomal RNA to study evolutionary spectrum of species.

- Given a DNA sequence that is highly homologous to some known tRNA gene, such a sequence may be a gene or a pseudogene. By predicting its secondary structure and examining its similarity to some tRNA secondary structure, one can verify whether or not it is a pseudogene.

4.1 Definition

In Section 2.2 we briefly discussed what RNA secondary structure is. Here, we present a mathematical definition as follows: The secondary structure of an RNA sequence of length n is an undirected graph $G = (V, E)$, where $V = \{1, ..., n\}$, $E \subseteq V \times V$ such that [21]:

1. $(i, j) \in E \Longleftrightarrow (j, i) \in E$
2. $(\forall 1 \leq i \leq n)[(i, i + 1) \in E]$
3. For $1 \leq i \leq n$, there exists at most one $j \neq i \pm 1$ for which $(i, j) \in E$
4. If $1 \leq i < k < j \leq n, (i, j) \in E$ and $(k, l) \in E$, then $i \leq l \leq j$

In this definition, it is important to remember that an edge between i and j where $j \neq i \pm 1$ corresponds to a Watson-Crick or wobble base pair. The first rule simply states that the base pair relationship is symmetric. The second rule connects all the consecutive bases together. The bases inside the sequence are linked by covalent energies and they should not be confused with the hydrogen bonds between Watson-Crick or wobble base pairs. The third rule is the rule that defines the base pairs inside the secondary structure and implies that each base can at most be paired with one other base. Finally the last rule is used to prevent the occurrence of interleaved base pairs inside the secondary structure.

There is also a 1-1 correspondence between RNA secondary structure and balanced parenthesis expressions, where balancing corresponds to base pairings. For instance the parenthesis expression ((...)) can be written for the sequence ACCCCGU in which (1,7) and (2,6) are the base pairs. By having the above definition an interesting question arises: how many secondary structures are possible for a sequence of n nucleotides? For answering this question, first we can assume that each base has the possibility to be paired with any other base (rather than restricting ourselves to Watson-Crick and GU pairs). In [21], it is proved that for such a hypothetical case, if we define $S(n)$ to be the number of secondary structures on $\{1, ..., n\}$ and let $S(0) = 1$:

Theorem 4.1.1 $S(1) = S(2) = 1$ *and*

$$S(n + 1) = S(n) + S(n - 1) + \sum_{k=1}^{n-2} S(k)S(n - k - 1).$$

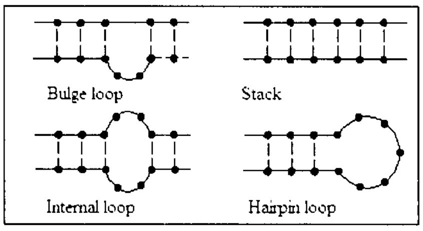


Figure 4.1: Some motifs inside RNA secondary structure

It is also proved that $S(n) \geq 2^{(n-2)}$, therefore there will be exponentially many possible secondary structures on a sequence of length n, when we ignore the base pair requirement. However, the exact asymptotic solution while considering these requirements is:

$$S(n) \sim \sqrt{\frac{15+7\sqrt{5}}{8\pi}} n^{-3/2} (\frac{3+\sqrt{5}}{2})^n$$

4.2 RNA motifs

Any RNA secondary structure (without pseudoknots) can be decomposed to a set of motifs as follows [21]:

1. The subsequence $i, ..., j$ in which every single base is unpaired is a *hairpin loop* if $i-1$ and $j+1$ are paired together
2. The subsequence $i, .., j$ in which every single base is unpaired is a *bulge loop* if $i-1$ and $j+1$ are also paired but not together.
3. If we have two bulge loops: $i, ..., j$ and $k, ..., l$ such that $i-1$ is paired with $l+1$ and $j+1$ is paired with $k-1$ and $i < j < k < l$ then these two bulge loops form an *interior loop*.
4. A *helix* is formed by adjacent base pairs stacking upon one another.
5. The subsequence $i, ..., j$ in which every single base is unpaired is a *single stranded region* if there is not a base pair enclosing this subsequence.
6. The subsequence $i, ..., j$ is called a *multi-loop* if i and j form a base pair enclosing other base pairs such as $k-l$ and $m-n$ which do not surround each other.

Some of these motifs are shown in Figure 4.1

4.2.1 Pseudoknots

Pseudoknots are one of the widely occurring structural motifs in RNA that play an important role regarding the functions of RNA. A simple pseudoknot is formed when unpaired bases from a loop form Watson-Crick base pairs with complementary bases outside the loop (Figure 4.2). Adding generalized pseudoknots to the problem of RNA secondary structure prediction makes it NP-hard. However, by using some heuristics and restricting the format of pseudoknots, this problem can be solved in polynomial time ($O(n^4)$) [28]. Most methods that predict RNA secondary structure, for simplicity, disregard the possibility of having pseudoknots. According to these methods, for every two base pairs: $pair(i, j)$ and $pair(i_1, j_1)$, if i_1 is greater than i then j_1 should be less than j. This assumption prevents the formation of pseudoknots in the result structure.

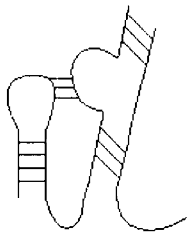


Figure 4.2: A simple pseudoknot

4.3 RNA Secondary Structure Prediction

Currently three major experimental methods are used to determine RNA secondary structure [15]:

- **Physical methods**, in which the structure is inferred base on the distance among atoms. These methods include X-Ray crystallography and Nuclear Magnetic Resonance (NMR).
- **Chemical methods**, where chemical and enzymatic probes are used to analyze the structure of RNA.
- **Mutation analysis**, which consists of initially mutating the RNA sequence and testing its binding ability with certain proteins.

Unfortunately these experimental methods have failed to generate cost and time efficient results. Meanwhile many bioinformaticians have constantly tried to solve this interesting while challenging problem using computational methods. Two or more limitations are usually assumed to simplify the problem of RNA secondary structure predictions. First, the most likely secondary structure is the energetically most stable structure. Second, the energy associated with any position is only influenced by local sequences and structure [43].

The bioinformaticians' contributions can be described within two main approaches. First, *minimum free energy* which is based on a single RNA sequence and finds the structure with the least free energy and second, *sequence comparison*, for which the idea is to align the sequence with the sequences with known secondary structure and predict the structure from the homology. Most work done to this day in both approaches assume that there are no pseudoknots in the secondary structure and the reason is, as mentioned earlier, that pseudoknots add much to the complexity of this problem. Although ignoring pseudoknots will reduce the accuracy of the method, since they are not very frequent, these methods can still give us good approximate results [15].

There also exist other methods which can not be categorized in these two approaches, however they are repeatedly being used to facilitate the primary methods. An example of them is *dot matrix sequence comparison* which is often used to find stretches of self-complementary regions inside sequences.

4.3.1 Minimum Free Energy

Finding the structure with considering only the minimum free energy is a very difficult and time consuming task for two reasons: first there might be many possible structures with the minimum free energy for a single sequence, and secondly, an accurate measure of the free energy is essential for this task. Over the past 30 years, another approach has been considered that simulates the thermal motion of RNA but since scientists lack complete knowledge regarding chemical and physical properties of the atoms, this results in approximate simulation of the energies and forces. Moreover as the approximate simulations even for short RNA sequences need faster CPUs to do the computations in a desirable amount of time, this approach is still quite inefficient [15].

All these efforts seemed hopeless until the *nearest neighbor model* was proposed by Tinoco et al. [15]. This relatively new model approximates the free energy of any RNA secondary structure using two prior assumptions: first, the energy of every loop is assumed to be independent of the others and second, the energy of a secondary structure is the sum of the energies of all the loops.

The long chains of stacked base pairs are the most stable secondary structures and they are the only possible structures that contribute negative free energy to the structure. Loops and bulges however increase the free energy in proportion to their size. As a result, a certain minimum number of stacked pairs is required to support a loop or bulge to the stacked pairs.

The first algorithm on this model using dynamic programming was proposed by Nussinov and Jacobson [48]. This algorithm tries to maximize the number of stacking pairs without considering the destabilizing energy of the loops. Following in 1981, Zuker and Steigler [66] introduced an algorithm which contemplates the destabilizing energies, however initially the time complexity was $O(n^4)$ but following that, in 1999 Zuker, Lyngoso, and Pedersen [41] improved the algorithm time complexity to $O(n^3)$ which was a major success and it is the best known algorithm so far. Zuker has also proposed a method to compute all the suboptimal structures in [63]. Eventually in 1991, Eppstein, Galil, and Giancarlo [29] proposed a new algorithm with a novel energy functions for loops and have improved the run time to $O(N^2 \log^2 N)$. The algorithm proposed by Zuker, Lyngoso, and Pederson is briefly explained below:

4.3.2 RNA Secondary Structure Algorithm without Pseudoknots

As mentioned above, loop energies play a very important role in predicting the secondary structure of RNA, therefore here we present the 4 energy functions derived by Mathews, Sabina, Zuker, and Turner that govern the formation of the loops [15]:

- $S(i,j)$ gives the free energy of the helix or stacked pairs consisting of pair (i,j) and $(i-1,j+1)$. This energy is negative and helps in stabilizing the structure
- $H(i,j)$ gives the free energy of a hairpin loop (pair (i,j) closes the hairpin loop and bases $i+1,...,j-1$ are unpaired inside the loop). This energy reduces the stability of the structure and it depends on $|j-i+1|$, i.e. length of the loop.
- $L(i,j,i_1,j_1)$ in which $i < i_1 < j_1 < j$, gives the free energy of an internal or a bulge loop. This loop is closed by pair (i,j) and (i_1,j_1) (if $i_1 = i+1$ or $j = j_1+1$ the loop is considered as a bulge loop). The free energy depends on the loop sizes $|i-i_1+1| + |j_1-j+1|$ and the asymmetry of the two loops.

- $M(i, j, i_1, j_1, ..., i_n, j_n)$ gives the free energy of a multi-loop where loops are closed by the pairs (i, j), (i_1, j_1), ..., (i_n, j_n).

Using dynamic programming and the assumption made in the nearest neighbor model which calculates the energy of a secondary structure as the sum of its loop energies, the energy of the optimal structure can be calculated as follows:

1. $V(i, j)$: the energy for the optimal secondary structure for the sequence $S[i, j]$ if (i, j) is a base pair:

$$V(i,j) = \begin{cases} \infty & \text{if } i \geq j, \\ min \begin{Bmatrix} H(i,j) & \text{Hairpin,} \\ S(i,j) + V(i+1, j-1) & \text{Stacked pair,} \\ VBI(i,j) & \text{Internal loop,} \\ VM(i,j) & \text{Multi-loop.} \end{Bmatrix} & \text{if } i < j. \end{cases}$$

2. $VBI(i, j)$: the energy for the optimal secondary structure for the sequence $S[i, j]$ if (i, j) and $(i_1, j_1)$are the base pairs enclosing a bulge loop or an internal loop.

$$VBI(i,j) = min_{i<i_1<j_1<j}\{L(i,j,i_1,j_1) + V(i_1,j_1)\}$$

3. $VM(i, j)$ the energy for the optimal secondary structure for the sequence $S[i, j]$ if (i, j) is a base pair that closes a multi-loop. The multi loop itself is formed by the base pairs$(i, j), (i_1, j_1), ..., (i_n, j_n)$ in which $n > 1$ and $i < i_1 < j_1 < ... < i_n < j_n$.

$$VM(i,j) = min_{i<i_1<j_1<...<i_n<j_n} \left\{ M(i,j,i_1,j_1,...,i_n,j_n) + \sum_{h=1}^{n} V(i_h, j_h) \right\}$$

4. $W(j)$: the energy of optimal secondary structure for the sequence $S[1, n]$ is given by the following recursive function

$$W(j) = \begin{cases} 0 & \text{if } i = 0, \\ min\{W(j-1), min_{1 \leq i < j}\{V(i,j) + W(i-1)\}\} & \text{when } j > 0. \end{cases}$$

4.3.3 Loop Dependent Energy Rules

In Section 4.2, we discussed different types of loops inside the RNA secondary structure. Below, we will show how the energy is calculated and assigned to all kinds of loops [65].

- **Hairpin loops:** according to polymer theory prediction, the free energy increment of a hairpin loop is calculated as:

$$\delta\delta G = 1.75 \times RT \times ln(l) \tag{4.1}$$

Here R is the universal gas constant, T is the absolute temperature and l is the length of the loop. In reality instead of using this formula, some tabulated values for $\delta\delta G$ for loops consisting of 3 to 30 nucleotides are used. For loops beyond size 30, the above equation is changed to:

$$\delta\delta G = \delta\delta G_{30} + 1.75 \times RT \times ln(l/30) \tag{4.2}$$

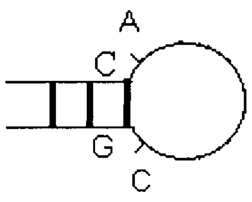
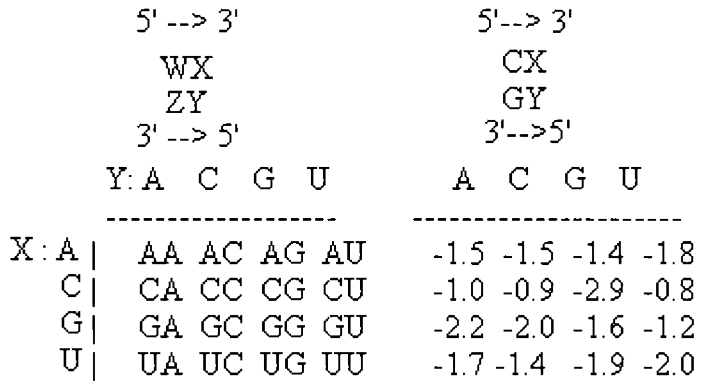


Figure 4.3: A hairpin loop

In addition to $\delta\delta G$ another factor is taken into account for hairpin loops of size greater than 3, which is the effect of *terminal mismatched pairs*. These parameters are also calculated and stored in several tables. One of the tables is shown and explained in Figure 4.4. Inside the right table, we can find the energy of a loop for which the closing base pair is CG. As an example, suppose we have a hairpin loop as shown in Figure 4.3, using the table in Figure 4.4, here A is assigned to X and C is assigned to Y and the corresponding energy for this hairpin loop is -1.5.

By using these tables, we treat all hairpin loops in the same way, which is proved to be not true in general. Some special rules apply to *triloops* (hairpin loops of size 3) and *tetraloops* (hairpin loops of size 4).



```
5' --> 3'        5'--> 3'
   WX               CX
   ZY               GY
3' --> 5'        3'-->5'
```

	Y: A	C	G	U
X : A	AA	AC	AG	AU
C	CA	CC	CG	CU
G	GA	GC	GG	GU
U	UA	UC	UG	UU

A	C	G	U
-1.5	-1.5	-1.4	-1.8
-1.0	-0.9	-2.9	-0.8
-2.2	-2.0	-1.6	-1.2
-1.7	-1.4	-1.9	-2.0

Figure 4.4: The left table shows a typical 4x4 table with the pairs WX and YZ covalently linked. WZ is the closing base pair of a hairpin loop while XY is a mismatched pair. The right table demonstrates the energies for terminal mismatched pairs when W is C and Z is G.

- **Stacks:** stacks refer to the stacking between the two immediately adjacent base pairs in a loop. Free energies for stacks are stored in 4x4 tables similar to the ones used for terminal mismatched. In Figure 4.5, a sample table is brought and briefly explained.

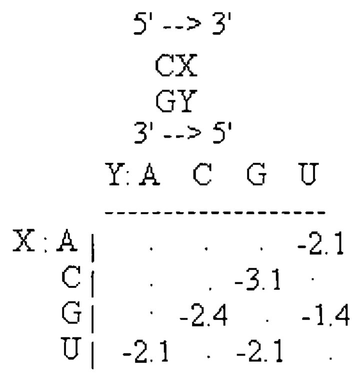
- **Bulge loops:** for bulge loops of size 1, the surrounding pairs are still treated as being stacked upon each other. For the ones up to size 30, the same tables used for hairpin loops are applied and for larger ones Equation 4.2 is used.

- **Interior loops:** for interior loops of size greater than 4, the free energy of the interior loop depends on 4 components:

$$\delta\delta G_{\text{I}} = \delta\delta G_{\text{I}}^{1} + \delta\delta G_{\text{I}}^{2} + \delta\delta G_{\text{I}}^{3} + \delta\delta G_{\text{I}}^{4} \tag{4.3}$$

  In Equation 4.3, $\delta\delta G_{\text{I}}^{1}$ is the same as $\delta\delta G$ in Equation 4.1 or 4.2, depending on the size. $\delta\delta G_{\text{I}}^{2}$ and $\delta\delta G_{\text{I}}^{3}$ are terminal mismatch free energy for the surrounding base pairs. For those interior loops which are *asymmetric* , a penalty component $\delta\delta G_{\text{I}}^{4}$ is considered.

  For 1x1, 1x2, and 2x2 interior loops, instead of using Equation 4.3, the free energies are calculated and stored in a set of tables. A 4x4 sample table which is used for 1x1 interior loops is brought in Figure 4.6.



5' --> 3'
CX
GY
3' --> 5'

X \ Y	A	C	G	U
A	.	.	.	-2.1
C	.	.	-3.1	.
G	.	-2.4	.	-1.4
U	-2.1	.	-2.1	.

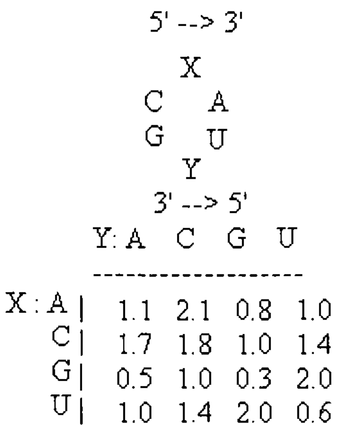
Figure 4.5: Sample free energies in Kcal/mole for CG base pairs stacked over all possible base pairs indicated by XY, where X refers to the row and Y refers to the column.

4.3.4 Sequence Comparison

Sequence comparison or sequence covariation model is the second major method for solving the problem of RNA secondary structure. Here, the goal is to find regions inside RNA sequences from the same RNA molecule of different species which are different literally but in reality they form the same structure. The idea behind this method is that although RNA sequences would be affected by evolutionary changes among species, these changes will not result in transforming the structure i.e. the changes would be made in such a way that the complementary base-pairings would be maintained [43]. For example, suppose we have an RNA sequence in which position 20 : C and position 41 : G form a base pair, according to this theory if the nucleotide in position 20 is transformed to A, the corresponding nucleotide in position 41 would also be changed to U, accordingly inside the new sequence, these two positions would still form the base pair. As we do not use the sequence comparison method in our system, we will not get into more details about this method.

4.4 RNA Secondary Structure Grammar

Clearly, it is very desirable to have a set of grammar rules for the genes, chromosomes, and even genomes. But unfortunately it is not an easy task to accomplish. Some of the grammar models written for genes are very complex and they are not guaranteed to recognize every possible gene. The grammar model of secondary structure of RNA is to some extent easier


5' --> 3'
X
C A
G U
Y
3' --> 5'

X \ Y	A	C	G	U
A	1.1	2.1	0.8	1.0
C	1.7	1.8	1.0	1.4
G	0.5	1.0	0.3	2.0
U	1.0	1.4	2.0	0.6

Figure 4.6: Free energies of a 1x1 interior loop closed by a CG and AU base pair.

and significantly smaller. For defining a grammar for RNA secondary structure, we first begin with RNA palindromes. As shown in Figure ?? the sequence AGCAUAUGCU is an RNA palindrome, i.e., the paired bases has the property of reading the same in either direction. RNA palindromes can have arbitrary lengths, therefore they need to be modeled by context-free grammars (or more complex ones). A grammar written for RNA palindromes is [6]:

$$s \rightarrow AsU|UsA|CsG|GsC|\varepsilon$$

The above sequence can be generated by: $s \rightarrow AsU \rightarrow AGsCU \rightarrow AGCsGCU \rightarrow AGCAsUGCU \rightarrow AGCAUAUGCU$. The RNA secondary structure palindromes are not always as perfect as this example. First of all, other base pairs such as UG or GU might occur in between (As mentioned in Section 2.2, base pairs like GA are also possible but quite rare). Secondly, regions of unpaired bases are also common. Loops usually should contain more than three or four bases because of the fact that RNA is not flexible enough to make a 180^o turn at the tip of the hairpin [6]. And finally, the palindromes inside RNA are usually more complex than the above example. One palindrome can be nested inside the other to make a *recursive palindrome* and two or more palindromes can follow each other side by side building up a *sequential palindrome*. Examples of recursive palindromes are *cloverleafs* inside tRNA. and different structures consisting of stems and loops inside rRNA. For adding simple recursive palindromes to the above grammar, one can easily employ $s \rightarrow ss$ but unfortunately this simple grammar rule makes the grammar ambiguous with alternative parse trees leading to alternative secondary structures. For example the sequence CAUG-CAUG can be derived by the first grammar rule generating a regular palindrome or by using the second grammar rule it can be decomposed into two palindromes. This ambiguity in structure is not surprising, in fact many examples of ambiguities can be found in DNA linguistics such as HIV viruses which benefit from overlapping genes [6].

Using the above grammar rules and adding a few more to include the unpaired bases inside palindromes, the final set of grammar rules for generating RNA secondary structure is [6]:

$$s \rightarrow AsU|UsA|CsG|GsC$$

$$s \rightarrow GsU|UsG$$

$$s \rightarrow As|Us|Cs|Gs|A|U|C|G$$

$$s \rightarrow ss$$

The first two rules are used to generate regular RNA palindromes (Watson-Crick and $G - U$ base pairs). The third rule, on the other hand, is used to insert the unpaired bases inside and outside palindromes forming different kinds of loops and single stranded regions. As mentioned above, RNA structure often happens to be more complex than simple palindromes surrounding loops. To be able to represent all these nested and recursive palindromes, rule number 4 is added to this set. Finally, although this grammar is proved ambiguous, we might actually prefer it to the non-ambiguous one for the reason that it is capable of modeling an underlying biological ambiguity, like the one shown above.

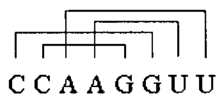


Figure 4.7: A sample pseudoknot

4.4.1 Pseudoknots Language

The grammar introduced above is only capable of generating secondary structures without pseudoknots. Features such as pseudoknots are referred to as *non-orthodox secondary structure.* Pseudoknots can be viewed as palindromes that are interleaved rather than nested

(Figure 4.7) and can be described using context-sensitive grammars. The formula $uvu^{-r}v^{-r}$ is usually used to define a simple pseudoknot language. The proposed grammar for this language by Searls [53] is:

$$S \rightarrow bS^{b}|A \quad A^{b} \rightarrow bA\bar{b}|B \quad B \rightarrow \bar{b}B \quad B \rightarrow \varepsilon$$

4.5 Summary

In this chapter we discussed the secondary structure of RNA in detail. We presented one of the famous definitions of RNA secondary structure in a graph where bases are shown as vertices and edges define the links between bases inside the secondary structure. We also introduced two general approaches for solving the problem of RNA secondary structure prediction. As an example, Zuker's algorithm for RNA secondary structure prediction was presented in detail. In the next chapter, we explain two state-of-the-art methods that are used to solve the inverse problem of RNA secondary structure prediction namely, RNA secondary structure design.

Chapter 5

Current Methods

There are currently two known methods to deal with the RNA secondary structure design problem. Both methods' objective is to find a sequence for which the minimum free energy (MFE) secondary structure matches the given structure. However, as mentioned in Chapter 4, the complexity of the best known algorithm for finding the secondary structure of RNA is $O(n^3)$ which would be a major drawback for both methods. Each method has a unique approach to overcome this problem.

RNA-SSD [5] is based on a stochastic local search (SLS) approach that uses a probabilistic sequence initialization heuristic, hierarchical decomposition of the given structure, and a randomized iterative improvement method for finding sequences for the resulting substructures. RNAinverse [33] also initializes the sequence but for each step it will change a base randomly and then compute the free energy for the result. In the following sections, we will explain these two existing methods in detail.

5.1 RNA-SSD: RNA Secondary Structure Designer

The RNA secondary structure design problem can be defined as follows: given an RNA secondary structure S^*, find a sequence X^* such that the minimum free energy structure for X^* matches S^*. Let Φ denote a function that assigns to each RNA sequence X, a secondary structure S^* that minimizes free energy over all possible secondary structures S of X. Now the problem can be formalized as finding a sequence X^* such that $\Phi(X^*) = S$.

There are two key components to RNA-SSD algorithm: stochastic local search (SLS) procedure and initial design. The core of the RNA-SSD algorithm is a stochastic local search

(SLS) procedure that iteratively modifies single unpaired bases or base-pairs of a candidate strand in order to obtain a sequence that folds into the desired structure. After each modification, the SLS algorithm makes a call to the RNA secondary structure prediction algorithm (with $O(n^3)$ complexity) to evaluate the new sequence. To reduce the number of calls to this algorithm, the input secondary structure and the initial sequence are hierarchically decomposed into substructures and at the lowest level, the SLS algorithm is applied only to those substructures. These partial solutions are then combined and form candidate solutions (not a final answer) for the larger subproblems. Considering that these partial solutions are not independent from each other and may result in totally different structure, some additional calls to the folding algorithm are often needed during the combination phase.

For each candidate solution X, S is calculated as $S = \Phi(X)$ and given a distance metric d which in this algorithm merely measures the number of incorrectly bonded bases, $d(S, S^*)$ is minimized.

5.1.1 Sequence Initialization

A good initial design is another key component for the RNA-SSD algorithm to support the goal of the next phase, i.e. minimizing the number of candidate solution evaluations. During the initialization phase, bases are assigned to the sequence by using different probabilistic models for paired and unpaired bases. Three insights are used in this phase to initialize the sequence in such a way that it is as close as possible to the MFE structure.

- The paired bases should be assigned in such a way that they form complementary base pairs.
- CG and GC base pairs are assigned with more probability to the base-pairs as they are energetically more favorable than other base pairs.
- To further minimize the potential for undesired interactions between subsequences, short sequences of bases (sequence motifs) are assigned to contiguous segments of the target structure.

Using these insights, the whole initialization phase can be explained as follows: initially, the structure is divided into chunks of successive paired or unpaired bases with the maximum size of l_{max} somewhat arbitrarily set to 10. Next, each segment is traversed from the 5' to the 3' end, and a sequence motif m is assigned. The probabilities for choosing each base for the motifs are p_A, p_C, p_G, p_U with $p_G = p_C, p_A = p_U = 1/2 - p_C$. For chunks of paired bases $p_C = 0.33$ is used while for the unpaired chunks p_C is set to 0.17.

5.1.2 Sequence Decomposition

For the phase of sequence decomposition, both RNAinverse and RNA-SSD split the structure at multiloops. However for RNA-SSD the split continues recursively to obtain a decomposition tree whose root is formed by the full target structure S and whose leaves correspond to small substructures of S. The whole process of sequence decomposition for the RNA-SSD algorithm can be explained as follows: the algorithm begins with the whole structure and continues decomposition as long as the structure to be decomposed is not smaller than a certain limit MaxSplit and both resulting substructures are not smaller than MinSplit. MaxSplit is set to 70, and MinSplit is set to 30.

5.1.3 Stochastic Local Search Procedure

The SLS procedure starts from the leaves of the decomposition tree (the smallest substructures) and iteratively modifies single bases of the subsequence initially assigned to that substructure which are in conflict with the desired structure. The iteration continues until either a subsequence is found for which the MFE structure corresponds to the substructure or a maximal number of iterations have been performed (5000 iterations). The sequences associated to the leaves are merged together to form a candidate sequence for the corresponding substructure.

5.2 RNAinverse

This algorithm works through an adaptive walk search on so-called compatible sequences, i.e. the sequences that can form a base pair at the required positions in the desired structure. A compatible sequence can but need not have the target structure as its minimum free energy structure.

The RNAinverse algorithm starts by choosing a random sequence I_0 and in each step, modifies it such as to minimize a cost function given by the structure distance: $f(I) = d(S(I), I)$ between the structure $S(I)$ of the sequence I and the target structure S^*. Similar to RNA-SSD, to avoid repeated calls to the cost function which results in many executions

Table 5.1: Performance results for RNA-SSD and RNAinverse on sets of artificial RNA structures. Each line represents the results of 100 RNA instances. The RNA-SSD and RNA-inverse columns show the fraction for which the respective algorithm found solutions. The columns, t(RNA-SSD) and t(RNA-SSD)*10 show the fraction of structures which RNA-SSD solve faster than RNA-SSD and at least 10 times faster respectively.

RNA Length	RNA-SSD	RNAinverse	t(RNA-SSD)	t(RNASSD)*10
178-447	100/100	63/100	100/100	57/100
164-465	85/100	5/100	83/86	73/86
172-409	88/100	3/100	90/92	81/92
203-402	88/100	2/100	81/84	72/84

of the folding algorithm (with expensive time complexity), the structure is partitioned to smaller substructures and they would be the first target for optimization. This method has two significant benefits: first the number of calls to the folding algorithm is substantially decreased and second, it reduces the probability of getting stuck in a local minimum.

As mentioned above, for the actual optimization phase, the adaptive walk is used. The adaptive walk will try a random mutation by exchanging one base that is not paired in the desired structure, or exchanging two paired bases with regards to the compatibility. The change is accepted if the cost function is reduced and rejected otherwise. The search steps continue until either a solution is found or a certain number of changes have been induced without any advantageous results. Finally, the algorithm concatenates the resulting subsequences to find the sequence corresponding to the full structure.

5.3 Comparison

As shown above, the major differences between RNA-SSD and RNAinverse are in sequence initialization, structure decomposition and sequence assembly as well as substructure search. The RNA-SSD algorithm is evaluated in [5] on randomly generated, computationally predicted structures of naturally occurring sequences from the Ribosomal Database Project (RDP), and on biological structures as well.

The results of the comparison between RNA-SSD and RNAinverse show that RNA-SSD substantially outperforms RNAinverse on a broad range of structures. Some of these results are shown in Table 5.1

Chapter 6

chr-statRNA

To solve the problem of RNA secondary structure design, we made use of the set of Context Free grammar rules for RNA secondary structure prediction introduced in Section 4.4 :

$$S \rightarrow cSg|gSc|aSu|uSa|gSu|uSg$$

$$S \rightarrow aS|gS|uS|cS$$

$$S \rightarrow a|g|u|c$$

$$S \rightarrow SS$$

This set of relatively simple rules presents a profound insight into the secondary structure of RNA and has been used as a basis for some RNA secondary structure prediction algorithms [6]. We use CHR to implement this grammar in order to exploit the bottom-up characteristic of CHR rules, as well as keep track of ambiguous readings with no special overhead.

At the first step, we translated these rules into the format of CHR rules, adding only one extra rule to this set which ascertains that the bases right after a loop would not be able to be paired together. In our system, the structure of the RNA (input data) is shown in the format of CHR constraints, e.g. for expressing that the base number 1 and the base number 43 in the sequence are paired together, we add the constraint `pair(1,43)` or if base number 3 is unpaired, the corresponding constraint would be `upair(3)`. One advantage of our input format to the input format used by RNAinverse and RNA-SSD is its capability of representing pseudoknots in the input structure.

After constructing the CHR rules, the problem becomes that of assigning nucleotides to each position given the input constraints. One trivial solution is to randomly assign one of the Watson-Crick pairs or wobble pairs to each base pair and assign one of the four nucleotides (A, C, G, and U) to the unpaired bases.

The problem with the random solution is that, although we follow the structure to build the sequence, there is no preference criteria to select between the existing pairs, so we might end up with a sequence that may not actually fold into the input structure. As mentioned earlier, the number of GC pairs has an important role in stabilizing a certain structure. For instance, if we assign base G to position 10 and base U to position 42 and if we have a base C in position 43, in the end, the structure might include `pair(10,43)` instead of `pair(10,42)` because of the fact that GC base pairs are stronger than GU pairs (Figure 6.1).

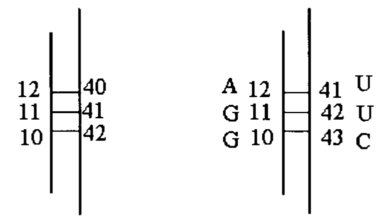


Figure 6.1: The left picture shows the original structure while the right picture illustrates the structure of the designed sequence.

Here, we offer two solutions to this problem: use of probabilities and use of energy tables. In the first solution which is the statistical version of chrRNA, namely chr-statRNA, we find the probabilities that are believed to govern the proportion of base pairs and single bases within RNA sequences and combine them with the CHR rules. The second approach which is the loop energies version of chrRNA, chr-loopRNA, exploits the energy tables that are currently used in several methods of RNA secondary structure prediction such as mfold [64, 42].

6.1 Use of Nucleotide Composition Probabilities

Algorithms for RNA secondary structure prediction that use a sequence comparison method design models of nucleotide frequency for each type of RNA (rRNA, tRNA, etc.) [17]. These models generally depict the frequency of each nucleotide at a certain position for unpaired bases and the frequency of the possible base pairs for the paired positions. However, our problem of interest, RNA secondary structure design, is a more general problem and the position of the nucleotides and the type of RNA may or may not be defined by the user. In our system, we are interested in finding the probabilities of having each single base and each base pair inside wide-ranging RNA.

As a result, we were not able to use the already made models. Instead we calculated the probabilities by comparing 200 RNA sequences of different kinds together. These sequences were selected from several databases of known RNA secondary structures such as Gutell lab's comparative RNA website [17], tmRNA website [32] and 5S Ribosomal RNA database [56]. After comparing 200 test cases with various lengths from 100 to 1500 bases, we found the following probabilities for each base pair:

$$P_{CG} = 0.53, P_{AU} = 0.35, P_{GU} = 0.12$$

The other set of probabilities which are of interest are the probabilities for an unpaired base to be one of the 4 nucleotides: A, C, G, or U. This set of probabilities includes all kinds of loops but it can also easily be divided into separate lists for each type of loop. The results are as follows:

$$P_G = 0.18, P_A = 0.34, P_C = 0.27, P_U = 0.21$$

There is a simple explanation for the above probabilities. Base G can be paired with both base C and base U and as the number of GC pairs is important for RNA stabilization, the probability of an unpaired base to be base G becomes smaller. However, base U is not as important as base G for stabilization but still can be paired with both base G and base A and that is why it has the second smallest probability to stay unpaired. Both base A and base C can only be paired with one nucleotide but as most base C tend to be paired with base G, the probability of an unpaired base to be base C would become less than base A.

6.1.1 Combining the Probabilities with the Rules

Mere translation of the RNA grammar into CHR rules is not enough for our purpose which is assigning nucleotides and base pairs based on certain probabilities. To achieve our goal, the probabilities need to be merged with the rules. Probabilistic Constraint Handling Rules [31] introduced by Frühwirth et. al. seems a reasonable solution to this problem, but this feature has not yet been added to current Prolog engines such as Sicstus, SWI, etc.

Fortunately, for our grammar rules we only need to consider the probabilities for the first two sets. The first set includes the rules which assign one of the 6 compatible pairs to a base pair while the second set contains four rules corresponding to the four possible nucleotides A, C, G, and U to be assigned to the single bases.

This enables us to merge each set of rules together to form a single rule where probabilities are involved by generating a random variable in the guard section of the rules, which is the only part that accepts Prolog predicates. Next, this random variable is compared to the above calculated probabilities: for instance for the following rule if the random variable `I` is less than 0.53, a GC pair is assigned to `pair(X1,Y1)`. The argument `L` in constraint `s/3` contains the list of bases already added to the sequence and the predicate `find/3` assigns a base pair to arguments `M` and `N` based on the random variable `I`.

```
pair(X1,Y1)\s(X,Y,L)<=>X1 is X-1,Y1 is Y+1,random(I),find(M,N,I)|
                          s(X1,Y1,[X1:M,Y1:N|L]).
```

Note that the probabilities are not directly calculated inside the rules. Instead, a random variable is generated and then the predicate `find/3` is called to find a base pair M, N according to the probabilities. In our algorithm we do not differentiate between a GC vs. CG pair. When we examine the random variable inside the rule, in order to assign the base pair, we make use of another predicate `choose/4` which chooses between the symmetric base pairs: GC or CG, AU or UA and GU or UG with a probability of 0.5 .

```
find(M,N,X):- X < 0.53,!,choose(g,c,M,N).
find(M,N,X):- X < 0.88,!,choose(a,u,M,N).
find(M,N,X):- choose(g,u,M,N).

choose(M,N,M,N):- random(I),I < 0.5,!.
choose(N,M,M,N).
```

Above, you can see the Prolog rules for predicates `find/3` and `choose/4`. The numbers inside the rules represent the cumulative probabilities.

As mentioned before, RNA-SSD and RNAinverse do not accept pseudoknots in the input structure. However, our implementation provides the capability of handling structures with pseudoknots through the following rule. This rule finds pairs of nucleotides which violate the third condition in the mathematical definition of RNA previously presented in Section 4.1 and separates them into two strings and at the same time, according to the probabilities, assigns them one of the possible base pairs.

```
pair(X,Y)\pair(X1,Y1)<=> X < X1, X1 < Y, Y1 > Y,random(I),
             find(M,N,I)| s(X1,X1,[X1:M]),s(Y1,Y1,[Y1:N]).
```

6.1.2 Rules

Primarily, the idea behind statistical chrRNA was to only apply the RNA grammar rules (which were adapted to CHR format) along with the probabilities. For improving the chrRNA system, we tried going beyond the basic RNA grammar rules by adding new rules to the system which were not easy to deduce at the beginning. They were actually found by comparing the secondary structure of the predicted sequence with the original structure. For adding these rules, new constraints are added to the system to give us the capability to distinguish between different conditions. Here, we go through the new rules as well as the old ones which have been changed partly to include new constraints, but first we shall explain the distinction between the two constraints `s/3` and `loop/3`. The constraint `loop/3` is used in this program to join the unpaired bases inside the structure in a loop format while the constraint `s/3` refers to any type of structures such as loops, stacked pairs and their combinations. For both constraints, only the first two parameters, i.e. the beginning and ending positions of the structure are shown in the corresponding figures.

- Rule 1, simply assigns a nucleotide to a single base using random variable `I` and the predicate `find/2` and adds the constraint `loop/3` to the constraint store. Moreover, a constraint `base/2` is also created which holds the position of the single base as well as the assigned nucleotide.

  ```
  @ Rule 1:
   upair(X)<=> random(I),find(M,I)| loop(X,X,[X:M]),base(X,M).
  ```

- Rule 2 is used to join two consecutive loops together.

  ```
  @ Rule 2:
   loop(X,Y1,L1), loop(X1,Y,L2)
   <=> X1 is Y1+1,append(L1,L2,L)| loop(X,Y,L).
  ```

- Through rule 3, a complete hairpin loop is formed. As indicated earlier, while designing the hairpin loops the two ends should be examined to ascertain that they won't form a base pair when the secondary structure is formed. The predicate `check_pair/3` takes two bases and changes one of them in case they can form a complimentary base pair.

```
@ Rule 3:
pair(X,Y),base(Y1,N1)\loop(X1,Y1,[X1:M1|L1]),base(X1,M1)
<=> X1 is X+1,Y is Y1+1,random(I),find(M,N,I),check_pair(N1,M1,M2)|
s(X,Y,[X:M,Y:N,X1:M2|L1]),base(X1,M2).
```

- Rule 4 is used to join a base pair to an already formed structure where the ends of the structure are paired together.

```
@ Rule 4:
pair(X,Y)\pair(X1,Y1),s(X1,Y1,L1)
<=> X1 is X+1,Y is Y1+1, random(I),find(M,N,I)| s(X,Y,[X:M,Y:N|L1]).
```

- Rule 5 is a variation of rule 4 where bases in positions `X1` and `Y1` are single. Just like rule 3, we must make sure the ends of the structure `s` are not matching.

```
@ Rule 5:
pair(X,Y),base(Y1,N1)\s(X1,Y1,[X1:M1|L1]),base(X1,M1)
<=> X1 is X+1, Y is Y1+1,random(I),find(M,N,I), check_pair(N1,M1,M2) |
s(X,Y,[X:M,Y:N,X1:M2|L1]),base(X1,M2).
```

- Rule 6 is also similar to rule 4 except that here `X1` is paired with another base in position `Z` rather than `Y1` where the base in the position right after `Z` is unpaired (position `Z1`). A base pair is assigned to `X,Y` in such a way that `Z1` would not be able to form a complementary pair with `X` (Figure 6.2). This rule is rewritten with `pair(X1,Z)` replaced with `pair(Z,Y1)` and in this case position `Y1` and position `Z1` are examined.

```
@ Rule 6:
pair(X,Y),pair(X1,Z),base(Z1,N2)\s(X1,Y1,L1)
<=> X1 is X+1, Y is Y1+1,Z=\=Y1, Z1 is Z+1,random(I),find(M,N,I),
check_pair(N2,M,M1,N1) | s(X,Y,[X:M1,Y:N1|L1]).
```

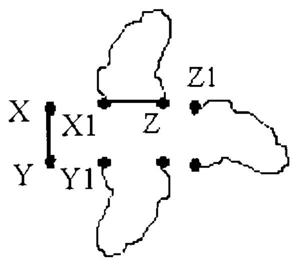


Figure 6.2: Rule 6

- Essentially, rule 7 is a variation of rule 6 when the base in position `Z1` is paired rather than unpaired. As a result, there is no need to check whether bases assigned to position `X` and position `Z1` are or are not able to form a complementary pair (Figure 6.3). Again, this rule is rewritten for the case that we have `pair(Z,Y1)` instead of `pair(X1,Z)`.

```
@ Rule 7:
pair(X,Y),pair(X1,Z),pair(Z1,T)\s(X1,Y1,L1)
<=>X1 is X+1, Y is Y1+1,Z=\=Y1, Z1 is Z+1,random(I),find(M,N,I)|
s(X,Y,[X:M,Y:N|L1]).
```

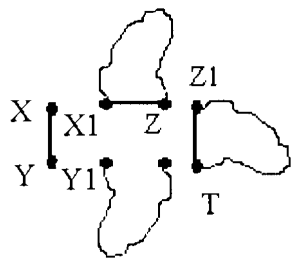


Figure 6.3: Rule 7

- Rule 8 corresponds to the rule $S \leftarrow SS$ which is used to join any two consecutive structures together. There is no probability involved and no special conditions need to be examined.

```
@ Rule 8:
s(X,Z,L1),s(Z1,Y,L2)<=>Z1 is Z+1, append(L1,L2,L) | s(X,Y,L).
```

- Rule 9 deals with bulge loops in general (Figure 6.4). This rule is rewritten for the case that the loop is adjacent to position Y.

```
@ Rule 9:
pair(X,Y),pair(Z1,Y1)\loop(X1,Z,L1),s(Z1,Y1,L2)
<=>X1 is X+1, Z1 is Z+1,Y is Y1+1,random(I),find(M,N,I),
append([X:M,Y:N|L1],L2,L) | s(X,Y,L).
```

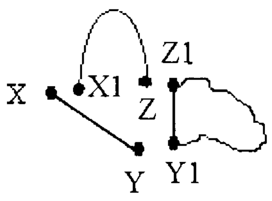


Figure 6.4: Rule 9

- Rule 10 addresses internal loops. Most of the required constraints are similar to the previous rule except that here two loops are involved. Using the predicate `check_pair/3` both ends of the two loops must be changed in a way that formation of complementary base pairs would be impossible for `Z-T1` and `X1-Y1` (Figure 6.5).

```
@ Rule 10:
pair(X,Y),pair(Z1,T),base(Z,M2),base(Y1,N1)\loop(X1,Z,[X1:M1|L1]),
loop(T1,Y1,[T1:N2|L3]),s(Z1,T,L2)
<=>X1 is X+1, Z1 is Z+1, T1 is T+1, Y is Y1+1, random(I), find(M,N,I),
check_pair(N1,M1,M3),check_pair(M2,N2,N3),append([X:M,Y:N,X1:M3|L1],L2,L4),
append(L4,[T1:N3|L3],L) | s(X,Y,L).
```

- To form a bulge loop we make use of rule 9. However if the endings of the structure `s/3` do not form a base pair together we make use of rule 11 (Figure 6.6). This rule is also rewritten for the symmetric condition.

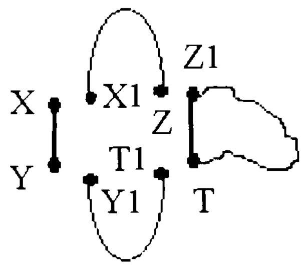


Figure 6.5: Rule 10

```
@ Rule 11:
pair(X,Y),pair(Z1,T)\loop(X1,Z,L1),s(Z1,Y1,L2)
<=>X1 is X+1, Z1 is Z+1,Y is Y1+1,T=\=Y1,random(I),find(M,N,I),
append([X:M,Y:N|L1],L2,L) | s(X,Y,L).
```

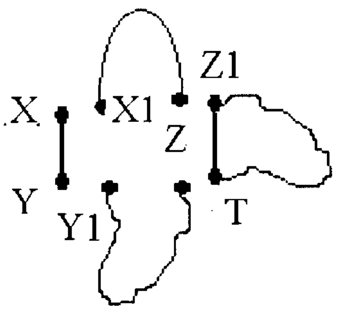


Figure 6.6: Rule 11

The above set of rules, almost considers all different substructures that might be found inside the secondary structure of RNA. There are some other rules in the final system which are not very significant but still are needed for the whole system to function properly. In the next chapter, we will discuss our second approach, chr-loopRNA, which solves the RNA secondary structure design problem using energies assigned to loops for calculating the probabilities. Finally, we will compare the results of both approaches with RNA-SSD and RNAinverse.

Chapter 7

chr-loopRNA

As previously mentioned, Zuker's method for RNA secondary structure prediction with energy minimization is one of the most practical and efficient methods in this field with $O(n^3)$ complexity. This algorithm was explained in detail in Section 4.3.2. In Section 4.3.3, we discussed the loop dependent energy rules and their corresponding energy tables used in Zuker's algorithm to calculate the energy for all possible kinds of loops inside the RNA secondary structure.

In our system, we do all the calculations beforehand and use the final results inside the rules. The major difference between the two versions of chrRNA is that for chr-statRNA, the probabilities were driven from the sequences themselves while in chr-loopRNA, we use the loop energy tables used by the mfold algorithm. It might be argued that using these energy tables makes chr-loopRNA dependent on the folding algorithm used for evaluation and makes the whole algorithm biased. Although we use these energy tables to find the required probabilities, the whole structure of the rules does not change if we change the probabilities. In the future, if a more efficient folding algorithm is developed, we can switch to this new algorithm by simply changing the probabilities inside the corresponding Prolog rules (see Section 6.1.1).

7.1 Use of Energy Tables

For each energy table we calculated a set of probabilities by using the following formula:

$$P(X_i) = \frac{|E(X_i)|}{\sum_{j=1..n} |E(X_j)|} \tag{7.1}$$

Table 7.1: The probabilities for stacking base pairs. The entry (i, j) is the probability of the base pair in row i being stacked on top of the base pair in column j.

X\Y	AU	CG	GC	GU	UA	UG)
AU	0.10	0.14	0.16	0.12	0.13	0.15
CG	0.23	0.22	0.18	0.28	0.25	0.23
GC	0.27	0.22	0.25	0.30	0.27	0.27
GU	0.14	0.16	0.16	0.10	0.17	0.14
UA	0.14	0.16	0.16	0.20	0.11	0.14
UG	0.12	0.10	0.09	0.00	0.07	0.07

X_i corresponds to the entries in the particular energy table whose corresponding probabilities we are interested in. Note that the energies inside the tables are mostly negative, so to calculate the probabilities we should consider the absolute value for $E(X_i)$. Next we shall explain the different types of energy tables and the consequent results for each table.

7.1.1 Stacking Energies

In Table 7.1, you can see the probabilities for stacking base pairs, calculated using the mfold energy tables for stacking energies. One of these tables was shown in Figure 4.5. Each probability (i, j) inside the table represents the probability of having the base pair in row i on top of the base pair in column j. As there are so many entries in this table (and the following tables), we used a heuristic to reduce the number of probabilities and consequently reduce the total number of required Prolog rules. This heuristic is based on the fact that the whole structure becomes more stable if we lower the energy of the whole sequence folded onto that structure. Here we do not calculate the whole energy to be able to reduce it, but at each point by having the base pair in column j, we can choose the base pair in row i that gives us the lowest energy (higher probability). Some of the values for probabilities are quite close to each other and for this reason we can not exclude all except the highest. For this reason, we only excluded the ones lower than a certain threshold $\alpha = 0.1$. The probabilities were then normalized on this new set.

Table 7.2: The probabilities for terminal mismatch of hairpin loops. Each entry (i, j) refers to the probability of having the base pair in row i as the closing base pair of a loop which begins with the incompatible pair in column j.

X\Y	AA	AC	AG	CA	CC	CU	GA	GG	UC	UU
AU	0.08	0.11	0.07	0.04	0.09	0.12	0.13	0.04	0.09	0.15
CG	0.38	0.33	0.31	0.37	0.41	0.47	0.25	0.36	0.44	0.27
GC	0.28	0.33	0.29	0.41	0.32	0.29	0.27	0.31	0.31	0.21
GU	0.00	0.11	0.07	0.04	0.09	0.12	0.10	0.07	0.09	0.15
UA	0.13	0.07	0.13	0.07	0.05	0.00	0.16	0.16	0.03	0.11
UG	0.13	0.05	0.13	0.07	0.04	0.00	0.09	0.06	0.04	0.11

7.1.2 Terminal Mismatch Stacking Energies (Hairpin Loops)

For hairpin loops, the closing base pairs play an important role in stabilizing the structure. As a result for calculating the energy of RNA secondary structures, the mfold algorithm makes use of a set of terminal mismatch stacking energy tables which are specifically generated for the closing base pairs of hairpin loops. Using these energy tables we calculated the probabilities using Equation 7.1. These probabilities are displayed in Table 7.2 with 6 rows corresponding to the 6 possible base pairs and 10 columns which are the remaining permutations of two nucleotides together (not forming base pairs).

Here, we only calculated the probabilities while there is a legitimate base pair closing a hairpin loop. Yet, the mfold algorithm should be able to calculate the energy for every possible structure. As a result there are also energy tables for the case in which a non compatible pair is closing a hairpin loop. Such energy tables are filled with 0's and do not have any effect in our calculations. For the probabilities of this table, the threshold α has been set to 0.1 and the remaining probabilities have been normalized respectively.

7.1.3 Terminal Mismatch Stacking Energies (Interior loops)

The closing base pairs of interior loops play the same role as the closing base pairs of the hairpin loops. However, because of the difference between the structure and the fact that there are two closing base pairs for every interior loop, these energies are different, therefore the resulting probabilities would be different. This set of probabilities is shown in Table 7.3. Once again, the threshold α is used to reduce the number of probabilities and the rules.

Table 7.3: The probabilities for terminal mismatch of interior loops. (i, j) represents the probability of having the base pair in row i as the closing base pair of an interior loop which begins with the incompatible pairs in column j.

X\Y	AA	AC	AG	CA	CC	CU	GA	GG	UC	UU
AU	0.00	0.00	0.11	0.00	0.00	0.00	0.11	0.00	0.00	0.05
CG	0.50	0.08	0.29	0.50	0.50	0.50	0.29	0.50	0.50	0.40
GC	0.50	0.08	0.29	0.50	0.50	0.50	0.29	0.50	0.50	0.40
GU	0.00	0.00	0.11	0.00	0.00	0.00	0.11	0.00	0.00	0.05
UA	0.00	0.42	0.11	0.00	0.00	0.00	0.11	0.00	0.00	0.05
UG	0.00	0.42	0.09	0.00	0.00	0.00	0.09	0.00	0.00	0.05

7.1.4 1x1 Interior Loop Energies

Above, we calculated the probabilities for the closing base pairs of interior loops. However, inside the interior loops there are other energy forces involved which influence the structure. These energies are calculated based on the size of the loops and the surrounding base pairs. For 1x1 interior loops, i.e the symmetric interior loops with only two incompatible bases surrounded by two closing base pairs, the resulting probabilities were 1 for almost all pairs of base pairs except for the two pairs of CG, CG and GC, GC. In Table 7.4, the probabilities are calculated for 1x1 interior loops with those surrounding base pairs. For the remaining 34 pairs of base pairs, the only pair of incompatible bases which result in negative energy is GG which is selected with the probability of 1.

Table 7.4: The probabilities for single mismatch energies inside 1x1 interior loops with CG-CG and GC-GC as the two surrounding base pairs.

X\Y	CG,CG	GC,GC
AC	0.16	0.00
AG	0.00	0.04
CA	0.00	0.16
GA	0.04	0.00
GG	0.68	0.68
UU	0.12	0.12

7.1.5 1x2 Interior Loop Energies

1x2 interior loops are asymmetric loops with one base on one side of the loop and two bases on the other side, surrounded by two closing base pairs. Inside the energy tables used by the mfold algorithm, one of the bases inside the loop (from the side with two bases) and the surrounding base pairs are assumed constants and the energies are calculated for having a pair of incompatible bases in the remaining sites. Unfortunately the energies inside these tables are all positive and we can not calculate the probabilities like before, by considering the ones with negative energy. Again we decided on another threshold $\beta = 3$ to eliminate some of the possible pairs and reduce the number of choices. As a result, any entry for which the energy is above β was excluded from the set of possible pairs. For most of the tables, this threshold restricted our domain to only one possible incompatible pair (Table 7.5).

Table 7.5: For the entries identified by x, whenever the respective base pairs on the row and the column surround a 1x2 interior loop, if one of the bases inside the 2 base loop is chosen to be A, we choose the pair GG as the remaining bases for the loop, otherwise if it is either U or C, we choose the pair UU.

	AU	CG	GC	GU	UA	UG
AU	x			x	x	x
CG						
GC						
GU	x			x	x	x
UA	x			x	x	x
UG	x			x	x	x

For the rest of the entries, the probabilities are calculated in Table 7.6-7.10. Once again to reduce the number of probabilities to deal with, the threshold α is used and the probabilities are normalized once more.

7.1.6 2x2 Interior Loop Energies

2x2 interior loops are symmetric interior loops with two surrounding base pairs and two bases in each side of the loop where each base on a side of the loop forms a mismatch pair with the opposite base on the other side. Considering that we have 6 base pairs and 10 mismatch pairs, the number of energy values and the resulting probabilities for 2x2 interior

Table 7.6: The probabilities for 1x2 interior loops with the following pairs of surrounding base pairs: AU-CG, AU-GC, CG-AU,CG-UA, CG-GU, CG-UG, GC-AU, GC-UA, GC-GU, GC-UG, UA-CG, UA-GC, GU-CG, GU-GC, UG-CG, and UG-GC. The bases in Y represent one of the bases in the 2-base loop and the mismatch pairs in X are the remaining bases inside the loop.

X\Y	A	C	G	U
AA	0.00	0.05	0.20	0.00
AC	0.00	0.24	0.27	0.00
AG	0.24	0.29	0.30	0.00
CC	0.00	0.00	0.00	0.17
GA	0.20	0.00	0.23	0.00
GG	0.56	0.00	0.00	0.00
UC	0.00	0.00	0.00	0.28
UU	0.00	0.42	0.00	0.55

loops are $6 \times 6 \times 10 \times 10 = 3600$. But the good news is that many of these values are positive and only a few of them are negative which are the ones that we use to calculate the probabilities. This on average would shrink the tables to 1/20 of their original size. Once again, we use the same threshold α to reduce the number of entries even more. The final results for the probabilities are shown in Table 7.11-7.16 .

7.1.7 Tetra-loops Energies

To calculate the energy of a hairpin loop (less than 30 nucleotides), Equation 4.1 is used but there are some exceptions where a bonus energy is also considered to calculate the whole energy. These exceptions include tri-loops and tetra-loops. The bonus energies for tetra-loops used by the mfold algorithm are available and we used them to calculate the probabilities for such loops (Table 7.17).

Table 7.7: The probabilities for 1x2 interior loops while the surrounding pair is CG-CG. The bases in Y represent one of the bases in the 2-base loop and the mismatch pairs in X are the remaining bases inside the loop

X\Y	A	C	G	U
AA	0.08	0.11	0.22	0.00
AC	0.09	0.16	0.24	0.00
AG	0.21	0.18	0.25	0.00
CA	0.08	0.10	0.00	0.09
CC	0.09	0.06	0.00	0.20
CU	0.09	0.10	0.00	0.15
GA	0.14	0.00	0.22	0.00
GG	0.22	0.00	0.07	0.28
UC	0.00	0.10	0.00	0.24
UU	0.00	0.19	0.00	0.32

Table 7.8: The probabilities for 1x2 interior loops where the closing base pairs of the loops could be either CG-GC or GC-CG. The bases in Y represent one of the bases in the 2-base loop and the mismatch pairs in X are the remaining bases inside the loop

X\Y	A	C	G	U
AA	0.07	0.11	0.20	0.00
AC	0.09	0.16	0.24	0.00
AG	0.16	0.17	0.26	0.00
CA	0.08	0.10	0.00	0.09
CC	0.09	0.06	0.00	0.20
CU	0.09	0.10	0.00	0.15
GA	0.15	0.00	0.22	0.00
GG	0.27	0.00	0.08	0.00
UC	0.00	0.10	0.00	0.24
UU	0.00	0.20	0.00	0.32

Table 7.9: The probabilities of 1x2 intereior loops where the surrounding pair is GC-GC. The bases in Y represent one of the bases in the 2-base loop and the mismatch pairs in X are the remaining bases inside the loop

X\Y	A	C	G	U
AA	0.06	0.11	0.18	0.00
AC	0.10	0.16	0.24	0.00
AG	0.11	0.17	0.27	0.00
CA	0.09	0.10	0.00	0.09
CC	0.10	0.06	0.00	0.20
CU	0.10	0.10	0.00	0.15
GA	0.16	0.00	0.22	0.00
GG	0.28	0.00	0.09	0.28
UC	0.00	0.10	0.00	0.24
UU	0.00	0.20	0.00	0.32

Table 7.10: The probabilities for 1x2 interior loops where the closing base pairs are: AU-AU,AU-UA, AU-GU, AU-UG, UA-AU, UA-UA, UA-GU, UA-UG, GU-AU, GU-UA, GU-GU ,GU-UG, UG-AU, UG-UA, UG-GU, and UG-UG. The bases in Y represent one of the bases in the 2-base loop and the mismatch pairs in X are the remaining bases inside the loop

X\Y	A	C	G	U
AA	0.00	0.00	0.16	0.00
AC	0.00	0.00	0.28	0.00
AG	0.00	0.00	0.34	0.00
GA	0.00	0.00	0.22	0.00
GG	1.00	0.00	0.00	0.00
UU	0.00	1.00	0.00	1.00

Table 7.11: The probabilities of having AU as one closing base pair while the other pair is determined by the base pairs in the first row (labeled Y). The two pairs in column one (labeled X) are mismatch pairs inside the 2x2 interior loops.

X\Y	AU	CG	GC	UA	GU	UG
AC,GU	0.00	0.12	0.17	0.00	0.00	0.00
AG,AG	0.00	0.11	0.00	0.00	0.00	0.00
CA,GU	0.00	0.13	0.19	0.00	0.00	0.00
GA,AG	0.00	0.11	0.00	0.00	0.00	0.00
GU,UG	0.00	0.20	0.00	0.00	0.00	0.00
UG,GU	1.00	0.34	0.64	1.00	1.00	1.00

Table 7.12: The probabilities of having CG as one closing base pair while the other pair is determined by the base pairs in the first row (labeled Y). The two pairs in column one (labeled X) are mismatch pairs inside the 2x2 interior loops.

X\Y	AU	CG	GC	UA	GU	UG
AC,GU	0.00	0.00	0.10	0.00	0.00	0.00
AG,AG	0.00	0.00	0.00	0.00	0.00	0.00
CA,GU	0.16	0.00	0.14	0.00	0.16	0.00
GU,UG	0.00	0.35	0.10	0.00	0.00	0.00
UG,AC	0.20	0.00	0.10	0.19	0.20	0.19
UG,CA	0.00	0.00	0.10	0.17	0.00	0.17
UG,GU	0.64	0.65	0.40	0.64	0.64	0.64

Table 7.13: The probabilities of having GC as one closing base pair while the other pair is determined by the base pairs in the first row (labeled Y). The two pairs in column one (labeled X) are mismatch pairs inside the 2x2 interior loops.

X\Y	AU	CG	GC	UA	GU	UG
AC,GU	0.00	0.13	0.00	0.00	0.00	0.00
GA,AG	0.00	0.17	0.00	0.11	0.00	0.11
GA,GA	0.00	0.00	0.00	0.11	0.00	0.11
GU,UG	0.22	0.26	0.35	0.20	0.22	0.20
UG,AC	0.15	0.13	0.00	0.12	0.15	0.13
UG,CA	0.00	0.00	0.00	0.00	0.00	0.12
UG,GU	0.38	0.31	0.65	0.34	0.38	0.34
UG,UG	0.25	0.00	0.00	0.00	0.25	0.00

Table 7.14: The probabilities of having UA as one closing base pair while the other pair is determined by the base pairs in the first row (labeled Y). The two pairs in column one (labeled X) are mismatch pairs inside the 2x2 interior loops.

X\Y	AU	CG	GC	UA	GU	UG
AC,GU	0.00	0.00	0.10	0.00	0.00	0.00
AG,AG	0.00	0.00	0.00	0.00	0.00	0.00
CA,GU	0.13	0.20	0.20	0.00	0.13	0.00
CA,UG	0.00	0.15	0.00	0.00	0.00	0.00
GU,UG	0.13	0.00	0.00	0.00	0.00	0.00
UG,AC	0.00	0.00	0.16	0.00	0.13	0.00
UG,CA	0.00	0.13	0.00	0.00	0.00	0.00
UG,GU	0.74	0.52	0.64	1.00	0.74	1.00

Table 7.15: The probabilities of having GU as one closing base pair while the other pair is determined by the base pairs in the first row (labeled Y). The two pairs in column one (labeled X) are mismatch pairs inside the 2x2 interior loops.

X\Y	AU	CG	GC	UA	GU	UG
AC,GU	0.00	0.00	0.17	0.00	0.00	0.00
CA,GU	0.00	0.19	0.19	0.00	0.00	0.00
GU,UG	0.00	0.30	0.00	0.00	0.00	0.00
UG,GU	1.00	0.51	0.65	1.00	1.00	1.00

Table 7.16: The probabilities of having UG as one closing base pair while the other pair is determined by the base pairs in the first row (labeled Y). The two pairs in column one (labeled X) are mismatch pairs inside the 2x2 interior loops.

X\Y	AU	CG	GC	UA	GU	UG
AC,GU	0.00	0.00	0.00	0.00	0.00	0.00
AG,AG	0.00	0.00	0.00	0.00	0.00	0.00
CA,GU	0.13	0.17	0.20	0.00	0.00	0.00
CA,UG	0.00	0.13	0.00	0.00	0.13	0.00
GU,UG	0.00	0.25	0.00	0.00	0.00	0.00
UG,AC	0.13	0.00	0.16	0.00	0.13	0.00
UG,CA	0.00	0.00	0.00	0.00	0.00	0.00
UG,GU	0.74	0.45	0.64	1.00	0.74	1.00

Table 7.17: The tetra-loops and their corresponding probabilities. The first and last bases represent the closing base pairs of the loops.

Tetra-Loop	Probability	Tetra-Loop	Probability
GGGGAC	0.05	GGUGAC	0.05
CGAAAG	0.05	GGAGAC	0.05
CGCAAG	0.05	GGAAAC	0.05
CGGAAG	0.05	CUUCGG	0.05
CGUGAG	0.05	CGAAGG	0.04
CUACGG	0.04	GGCAAC	0.04
CGCGAG	0.04	UGAGAG	0.04
CGAGAG	0.03	AGAAAU	0.03
CGUAAG	0.03	CUAACG	0.03
UGAAAG	0.03	GGAAGC	0.02
GGGAAC	0.02	UGAAAA	0.02
AGCAAU	0.02	AGUAAU	0.02
CGGGAG	0.02	AGUGAU	0.02
GGCGAC	0.02	GGGAGC	0.02
GUGAAC	0.02	UGGAAA	0.02

7.2 Designing the Loops

We discussed the energy tables and the probabilities used for assigning single bases as well as base pairs inside certain kinds of loops. However, there are still some positions inside most of the secondary structures which are not covered. Specifically, except for 1x1, 1x2, and 2x2 interior loops, there are no energies assigned to loops of different sizes. Zuker's algorithm uses the energy rules introduced in Section 4.3.2 for each type of loop which usually only consider the length of the loop and does not differentiate between different bases and their arrangements.

For RNA secondary structure design, we have the freedom to choose every single base, but we should be careful with the selections we make. For example, suppose we are designing a 15-base hairpin loop. If we assign G to one position and assign C to the position next to it and after passing about 4 nucleotides we assign nucleotide G and C adjacent to each other, the chances are that when the secondary structure is forming, the hairpin loop will not form like the original structure. Instead these four bases will form two adjacent base pairs and change the original hairpin loop structure into an internal loop and a smaller hairpin loop (Figure 7.1).

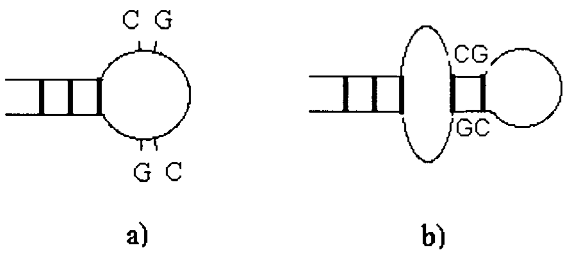


Figure 7.1: In a), a hairpin loop is shown with CG assigned to two pairs of two adjacent bases. In b), the structure of the same sequence in reality is illustrated.

7.2.1 A Heuristic

To overcome such problems while designing the loops, we use a heuristic to make sure we do not assign to consecutive bases, nucleotides that might form strong base pair combinations. To find such combinations, we studied the energy tables for stacking base pairs. These energy tables present the energies assigned to all combinations of adjacent base pairs. Using these, we came up with the best choices to assign to the adjacent position of every single base.

Again, some of the values are quite close for some combinations, therefore we calculated the probabilities and eliminated the ones less than a certain threshold α. The results are shown in Table 7.18.

Here, as energies are mostly negative and the ones with the lowest energies are favorable, we use the following formula to calculate the probabilities. As the values are all greater than -3.00, we choose $T = 3.00$ to add to the energy values:

$$P(X_i) = \frac{T + E(X_i)}{\sum_{j=1..n}(T + E(X_j))} \tag{7.2}$$

Table 7.18: In this table, any (i, j) entry represents the probability of having the base in column j right after the base in row i inside the RNA sequence when it is read from 3' end to 5' end.

X\Y	A	C	G	U
A	0.36	0.16	0.29	0.19
C	0.44	0.00	0.00	0.56
G	0.34	0.23	0.00	0.43
U	0.40	0.19	0.41	0.00

If we change the value of T to 4 in the the above equation, all the resulting probabilities would be above zero (although very small in some cases). This refinement would ascertain that we do not completely remove the chance of having every two consecutive bases. But on the other hand it might increase the chance that the loops in the final structure do not exactly match the given structure (Figure 7.1).

7.3 Adding the Probabilities to the Rules

The probabilities we obtained above can not be combined with the limited set of RNA grammar rules introduced in Section 4.4, therefore we have devised new rules to incorporate each set of probabilities such as: stacking loops, terminal mismatch stacking in hairpin loops and interior loops, etc. Below, you can see this set of new rules.

- To assign bases to a generic loop, we use rule 1. The predicate `findpattern/3`, uses the probabilities in Table 7.18 as well as a random variable and the first element of the loop to find a nucleotide for position X which would be attached to the head of loop.

```
 @Rule 1:
upair(X),loop(X1,Y,[X1:N|L1])
<=>X1 is X+1, random(I), findpattern(N,M,I)|
loop(X,Y,[X:M,X1:N|L1]),base(X,M).
```

- In rule 2, the goal is to add a closing base pair to a hairpin loop and form the structure `s/3` which contains the two end positions as well as their respective assigned bases. Inside the guard, using the random variable `I`, the terminal mismatch pair closing the hairpin loop (`M,N`) and the probabilities from Table 7.2, the predicate `findhm/4`, generates the base pair `M1-N1` to assign to positions `X1` and `Y1`. In addition, `X` and `Y` are examined to make sure that the hairpin loop is not a tetra-loop.

```
@ Rule 2:
 pair(X1,Y1),base(Y,N)\loop(X,Y,[X:M|L])<=>X is X1+1, Y1 is Y+1,
 Y-X=\=3, random(I),findhm(M,N,M1,N1,I) | s(X1,Y1,[X1:M1,Y1:N1|L]).
```

- We use rule 2 to design any hairpin loops except tetra-loops for which rule 3 is used. Instead of using the predicate `findhm/5`, here we use the predicate `tetloop/4` that only takes a random variable `I` as well as the positions of the base pair and outputs the list of all the positions inside the loop and their assigned bases inside a list.

```
@ Rule 3:
 pair(X1,Y1)\loop(X,Y,_)<=>X is X1+1, Y1 is Y+1,
 Y is X+3, random(I),tetloop(X1,Y1,L,I)| s(X1,Y1,L).
```

- In rule 4 the goal is to place a new base pair on top of a stack of base pairs. To achieve this goal, bases are assigned to the new base pair in such a way that they conform to the probabilities of stacking base pairs (Table 7.1). Using random variable `I` and the terminal base pairs of the stack M-N, predicate findst/5 generates bases M1-N1 to be allocated to positions X1 and Y1 respectively.

```
@Rule 4:
pair(X1,Y1)\pair(X,Y),s(X,Y,[X:M,Y:N|L])<=> X is X1+1, Y1 is Y+1,
random(I),findst(M,N,M1,N1,I)| s(X1,Y1,[X1:M1,Y1:N1,X:M,Y:N|L]).
```

- Rules 5 to 8 examine the required constraints and design different types of interior loops (1x1, 1x2 and 2x2). Rule 5, specifically targets 1x1 interior loops. Using random variable I, a Watson-Crick base pair M1-N1 is assigned to the closing base pair X1 and Y1. Using the closing base pairs assigned to positions X,Y and X1,Y1, the predicate findint11/7 assigns a mismatch pair M3,N3 to the bases inside the loop (Figure 7.2).

```
@Rule 5:
pair(X1,Y1)\pair(X,Y),loop(Z,Z,_),loop(Z1,Z1,_),s(X,Y,[X:M,Y:N|L])
<=>X1 is Z-1, X is Z+1,Z1 is Y+1,Y1 is Z1+1,
random(I),find(M1,N1,I),random(J),findint11(M1,N1,M,N,M3,N3,J)|
s(X1,Y1,[X1:M1,Y1:N1,Z:M3,Z1:N3,X:M,Y:N|L]).
```

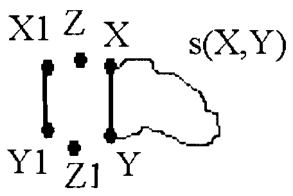


Figure 7.2: A 1x1 interior loop

- Rule 6, uses the probabilities calculated for 1x2 interior loops. In Figure 7.3, the conditions needed to form a 1x2 interior loop are illustrated. Here, using the constraints and the guard we make sure all the conditions hold. Moreover inside the guard, we initially assign a Watson-Crick base pair M1-N1 to position X1 and position Y1 which are the closing base pairs of this structure, through the predicate `find/3`. Next, using the newly assigned base pair, the base `N2`, the bases on top of the stack `M,N`, and the random variable `J`, the predicate `findint12/8` finds the corresponding mismatch pair for positions `Z` and `Z2` inside the loop.

```
@Rule 6:
pair(X1,Y1)\pair(X,Y),loop(Z,Z,_),loop(Z1,Z2,[Z1:N2,_]),
s(X,Y,[X:M,Y:N|L] )<=>X1 is Z-1, X is Z+1,Z1 is Y+1,
Y1 is Z2+1,Z2 is Z1+1,random(I),find(M1,N1,I),
random(J),findint12(M1,N1,M,N,N2,M3,N3,J)|
s(X1,Y1,[X1:M1,Y1:N1,Z:M3,Z1:N2,Z2:N3,X:M,Y:N|L]).
```

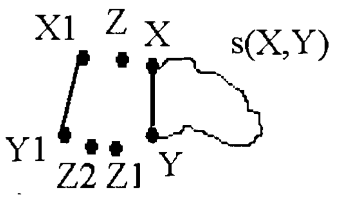


Figure 7.3: A 1x2 interior loop

- Similar to rule 5 and rule 6, rule 7 is also formulated to design an interior loop which in this case is 2x2. Once again, a Watson-Crick base pair is assigned to the base pair in positions `X` and `Y` and next the new base pair along with the closing base pair of the structure `s/3` and a random variable `J` are given as arguments to the predicate `findint22/9` which by using the corresponding probabilities for 2x2 interior loops, assigns the bases `M2,N2` and `M3,N3` to the positions `Z1,Z4,Z2,Z3` respectively (Figure 7.4).

```
@Rule 7:
pair(X1,Y1)\pair(X,Y),loop(Z1,Z2,_),loop(Z3,Z4,,_),
s(X,Y,[X:M,Y:N|L]) <=>X1 is Z1-1, X is Z2+1,Z3 is Y+1,
Y1 is Z4+1,Z2 is Z1+1,Z4 is Z3+1,random(I),find(M1,N1,I),
```

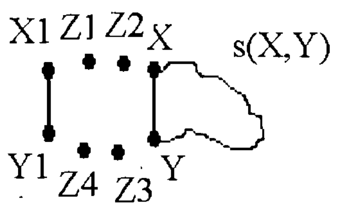


Figure 7.4: A 2x2 interior loop

```
random(J),findint22(M1,N1,M,N,M2,N2,M3,N3,J)|
s(X1,Y1,[X1:M1,Y1:N1,Z1:M2,Z2:M3,Z3:N3,Z4:N2,X:M,Y:N|L]).
```

- For the rest of the interior loops that do not fall into any of the above categories we use rule 8 (Figure 7.5). Here, we only make use of Table 7.3 to find a base pair to assign to the `X1,Y1` pair, but first we make sure that the terminal pairs of the interior loop are incompatible pairs by using the predicate `check_pair/3`. Following that, the predicate `findmm/5` is called to find the closing base pair of the loop.

```
@Rule 8:
pair(X1,Y1),base(Z4,N2),base(Z2,M3)\base(Z1,M2),base(Z3,N3),pair(X,Y),
loop(Z1,Z2,[Z1:M2|L1]),loop(Z3,Z4,[Z3:N3|L2),s(X,Y,L)
<=>X1 is Z1-1, X is Z2+1,Z3 is Y+1,Y1 is Z4+1,Z4-Z3 >1,
Z2-Z1>1,check_pair(N2,M2,M4),random(I),findmm(M4,N2,M1,N1,I),
check_pair(M3,N3,N4),append([X1:M1,Y1:N1,Z1:M4|L1],L,L3),
append(L3,[Z3:N4|L2],L4)| s(X1,Y1,L4),base(Z1,M4),base(Z3,N4).
```

- For bulge loops composed of only one base, the energy is calculated as if the closing base pairs were stacked upon each other. As a result we use a separate rule that considers 1-base bulge loops and for all the others, we use the same rule as the one used for chr-statRNA:

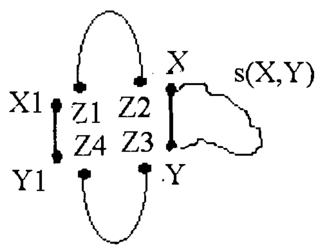


Figure 7.5: A generic interior loop

```
@Rule 9:
pair(X1,Y1)\pair(X,Y),loop(Z,Z,[Z:T]),s(X,Y,[X:M,Y:N|L])
<=>X1 is Z-1, X is Z+1,Y1 is Y+1,
random(I),findst(M,N,M1,N1,I)|
s(X1,Y1,[X1:M1,Y1:N1,Z:T,X:M,Y:N|L]).
```

Above, we introduced a set of rules used in chr-loopRNA. In the next chapter we will show the results of both chr-loopRNA and chr-statRNA.

Chapter 8

Results and Conclusion

For the evaluation phase, firstly we transform the secondary structure of each test case to constraint format by using a Java based program, namely `toConstraint.java` and then run the chrRNA program for each of those. Using the probabilities turns the result of each run on a test case into a completely new sequence. The sequence produced by chrRNA is then fed into an RNA folding server and the output is the structure of this new sequence. The two structures (the original structure and the output structure of the RNA folding server) are then compared with each other and the differences are marked. We have used the mfold RNA folding server by Zuker et. al. [64, 42]. The accuracy of this algorithm is estimated to be about 73% which shall be considered while evaluating the system. A schema of the whole evaluation system is shown in Figure 8.1

8.1 Comparisons

We have tested chr-statRNA and chr-loopRNA programs with 100 RNA secondary structures (without pseudoknots) from Gutell lab's comparative RNA website [17], tmRNA website [32] and 5S Ribosomal RNA database [56]. The test cases were selected from both Prokaryote (Archaea and Bacteria) and Eukaryote. The reason we only included RNA samples without pseudoknots in our test domain is that as mentioned before, there is still no efficient algorithm capable of predicting the secondary structure of RNA containing pseudoknots and therefore we are not yet able to assess chrRNA for designing sequences with pseudoknots.

For comparison, we have also evaluated RNA-SSD and RNAinverse using the same RNA

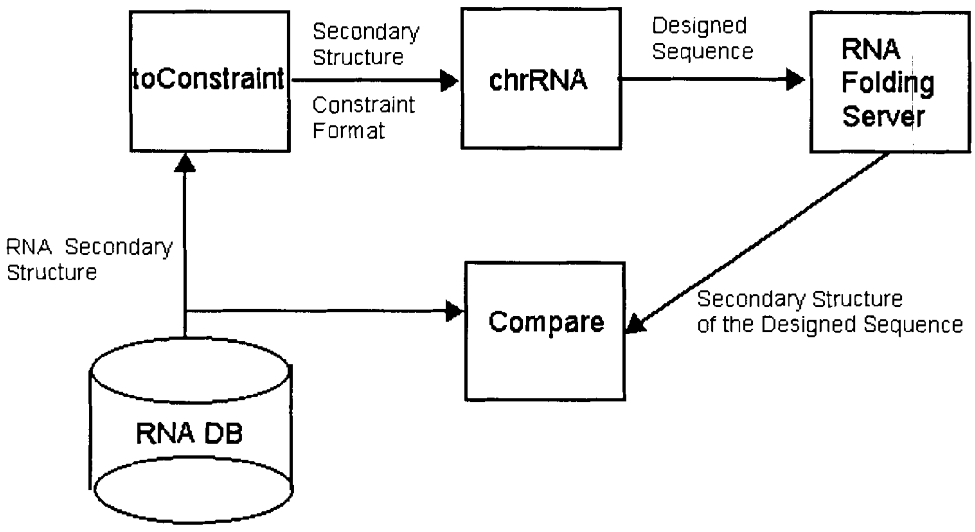


Figure 8.1: The Evaluation System

samples. RNAinverse is one of the programs inside the Vienna RNA package[1] [33]. RNAinverse and chrRNA were both executed on a Power Mac G5 with 1.5 GB memory and Dual 2 GHz PowerPC G5 CPU. RNA-SSD has been implemented [3] and is only available online[2] through a web application. We have divided our test cases into three groups according to their length. The results of our accuracy comparison between chr-statRNA, chr-loopRNA, RNA-SSD and RNAinverse are displayed in Table 8.1. The results of comparison between RNAinverse and chr-loopRNA (which is slightly more accurate than chr-statRNA) run time are shown in Figure 8.2.

The results show that for RNA sequences of less than 300 bases, RNA-SSD is the excellent choice with 0% error but this method is unable to find any solution for sequences longer than 300 base within their time-limit of 1 hour. For RNA sequences between 300 and 500 bases, the RNAinverse and chr-loopRNA error rates are quite close to each other, however as seen in Figure 8.2, chr-loopRNA is much faster in generating the results. For sequences consisting of more than 500 bases, the difference between chr-loopRNA and RNAinverse run-time becomes much more significant. While chr-loopRNA is able to design a 510 long

[1]http://www.tbi.univie.ac.at/~ivo/RNA/

[2]http://www.rnasoft.ca

Table 8.1: Comparison between chr-statRNA, chr-loopRNA, RNA-SSD, and RNAinverse. l defines the length of the RNA sequence and columns 2-5 show the average error of each method for 50 runs according to the length of the sequences.

Method	$l <= 300$	$300 < l <= 500$	$500 < l <= 1023$	$1023 < l <= 2000$
chr-statRNA	8%	22%	27%	35%
chr-loopRNA	6%	20%	22%	30%
RNAinverse	4%	18%	20%	–
RNA-SSD	0%	–	–	–

sequence in 60 seconds, RNAinverse needs 7200 seconds to finish the same job. The RNAinverse run-time for a sequence consisting of 700 bases is 4 hours with 22% error where the chr-loopRNA designs the same sequence in 13 minutes with 27% error. On the other hand, while RNAinverse does not accept sequences longer than 1023 bases, chr-loopRNA and chr-statRNA do not have any limitations on the length of the sequence.

8.2 Discussion

We implemented two simple while powerful systems based on CHR rules to design the RNA secondary structure: chr-statRNA and chr-loopRNA. Although both methods incorporate the same basic RNA grammar rules introduced in Section 4.4, each method extends this set of rules and combines them with probabilities differently. While chr-statRNA uses the nucleotide composition probabilities of RNA which are derived statistically, chr-loopRNA exploits the energy tables used in the mfold algorithm [64, 42] to calculate the probabilities to build up stacks and loops inside RNA. The results in Table 8.1 show that in general chr-loopRNA performs better than chr-statRNA. One reason that makes chr-loopRNA superior is that energies involved with the arrangements of nucleotides inside RNA are the same for all RNA types, but for chr-statRNA it might highly depend on which type of RNA we are designing. For instance it has been proved that GU pairs are more abundant inside tRNA molecules, so although the probability assigned to GU pairs is generally very low, for tRNA molecules this probability is higher.

Above we also show that although the two existing methods, RNA-SSD and RNAinverse, generate more precise results while designing shorter sequences (less than 300) by utilizing

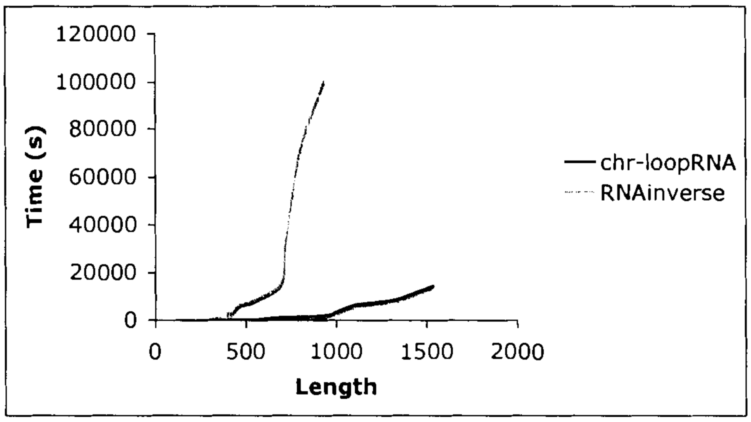


Figure 8.2: chr-loopRNA and RNAinverse run time in seconds

other non trivial rules such as energy rules into their methods which are not very easy to implement in our case of CHR rules; considering the execution time and accuracy factors together, chr-loopRNA performance is similar to RNAinverse for RNA sequences between the range of 300 and 500 bases. Finally, for RNA sequences longer than 500 bases, chr-loopRNA outperforms both methods by generating results in a much smaller amount of time. The reason for the large difference between chrRNA and RNAinverse response time is that the system implementing RNAinverse employs some RNA secondary structure prediction algorithms with $O(n^3)$ time complexity. Although in general using such algorithms to examine the designed sequences makes this method more accurate, the trade-off is a major increase in run-time for longer sequences. In addition to this, the capability of handling pseudoknots, makes our systems more powerful in comparison to the other existing methods. Further advantages of our implemented systems are:

- CHR rules are much more understandable by biologists, compared to the other programming languages.
- Using a rule-based system give us the opportunity to easily extend our system just by employing new rules. Biologists can even write their own rules and constraints and add them to the system.
- Our systems do not depend on RNA secondary structure algorithms with the expensive time complexity.
- The probabilities involved inside the rules can be easily changed according to biologists'

needs. For instance they can easily change the ratio of each nucleotide inside the designed RNA.

- chrRNA does not force any limitations regarding the length of the input structure.

8.3 Future Work

One future extension point for chr-statRNA is to calculate an individual set of probabilities and incorporate more samples for each type of RNA such as rRNA, tRNA, and etc. For chr-loopRNA we have already made use of all the state-of-the-art energy values for RNA secondary structures. In the future, if new energy rules between RNA nucleotides are discovered or if these energy tables are updated, the probabilities inside chr-loopRNA can be easily recalculated and adjusted as well.

Another improvement to our methods would consist in also incorporating more non-trivial rules which are not easy to predict, through some machine learning methods such as *Inductive Logic Programming* (ILP) [44]. ILP is a relatively new methodology of automatically eliciting hidden knowledge with the help of a computer [45]. ILP has already been used in molecular bioinformatics [52]. Some examples of this application include gene function prediction [36], protein secondary structure prediction [46] and drug design [35]. By using this methodology and adding new rules, this system could probably show better results in general.

Finally, it might be a good idea to somehow merge our method with RNA-SSD or RNAinverse. For instance, in the case of RNA-SSD, chrRNA can be used in the initialization phase to reduce the amount of differences between the predicted structure for the initially designed sequence and the original structure. By comparing the secondary structures of our designed RNA sequences and the given secondary structures, we have discovered that many of the inaccuracies appear while designing internal loops and long hairpin loops when adjacent single bases from one side of the loop form base pairs with bases of the other side. As seen in Section 7.2.1, we used a heuristic to reduce the chance of this happening inside the loop, but as we are dealing with probabilities here, there is no way to guarantee that this won't occur. While examining such conditions is not very efficient inside CHR rules, involving another program that employs energy rules and predicts the secondary structure for the designed loop, would be effective in generating more accurate results.

We expect the results presented here to be invaluable for in vitro genetics, by enabling scientists to produce RNA virtually from sequences; and for drug design, which typically progresses backwards from proteins to RNA to DNA.

Appendix A

Glossary of Biological Terms

Amino acid: basic structural building units of proteins.

Base pair: or complementary base pair refers to two nucleotides on opposite strands of DNA or RNA that are connected via hydrogen bonds.

Chromosomes: the separate physical molecules which are arranged to form DNA inside the cells.

Codon: each triplet of bases inside RNA.

Cytoplasm: the contents of a cell without the nucleus.

DNA (Deoxyribonucleic Acid): a nucleic acid that contains the genetic instructions specifying the biological development of all cellular forms of life.

Eukaryotes: organisms with complex cells such as animals, plants, and fungi.

Exon: the stretches of DNA that code for proteins.

Gene: the units of heredity in living organisms.

Genetics: the science of heredity in living organisms.

Genome: all the genetic information inside chromosomes.

Helix: a screw like structure.

Intron: the non-genetic stretches of DNA that separate exons in eukaryotes.

Nucleic acid: a biochemical macromolecule composed of nucleotide chains that convey genetic information. The most common nucleic acids DNA and RNA.

Nucleotide: the structural units of DNA and RNA which are Adenine(A), Cytosine(C), Guanine(G), Uracil(U, Thymine(T).

Prokaryotes: the organisms without a cell nucleus such as bacteria.

Proteins: the molecules responsible for much of the structure and activities of organisms that are composed of amino acids.

Protein folding: finding the tertiary structure of protein from its sequence of amino acids.

Pseudoknots: widely occurring structural motifs inside RNA that play important roles regarding the function of RNA.

RNA (Ribonucleic Acid): a chemical found in cells that codes for amino acids.

RNA folding: finding the structure of RNA from its sequence of nucleotides.

RNA primary structure: the sequence of nucleotides inside RNA.

RNA secondary structure: the information regarding the nucleotides inside the RNA sequence that shows which nucleotide bind to each other and which one remain single.

RNA secondary structure design: designing an RNA sequence that folds onto a given secondary structure.

RNA secondary structure prediction: predicting the secondary structure of an RNA sequence from its primary structure.

RNA splicing: the act of removing introns and joining exons.

RNA tertiary structure: the actual positions of molecules in three-dimensional space.

Splice sites: the boundaries of exons and introns.

Watson-Crick base pairs: the base pairs between C-G in DNA and RNA, A-T in DNA, and A-U in RNA.

Wobble base pairs: the base pairs between nucleotide G and U.

Appendix B

A Sample Run

In this appendix, we provide the information regarding the secondary structure of a sample RNA and we present a sample run of the program chr-loopRNA for this particular secondary structure. The sample secondary structure is:
.((((((((((....((((((((...(((((((.........))))..)))...)))))).)).(((((((..(((((((...)))))))..)))))))...)))))))))).

The result of a sample run on the above secondary structure is:

```
| ?- test8.
[1:a,2:c,119:g,3:g,118:c,4:c,117:g,5:g,116:c,6:u,115:a,7:c,114:g,
8:c,113:g,9:g,112:u,10:g,111:c,11:c,110:g,12:c,13:a,14:c,15:u,16:a,
70:g,17:g,69:c,18:g,68:c,19:g,66:c,20:a,65:u,21:g,64:c,22:c,63:g,
23:g,62:u,24:g,61:c,25:c,26:a,27:g,28:a,29:g,57:c,30:c,56:g,31:g,
55:c,53:a,54:a,32:g,52:c,33:g,51:c,34:u,50:g,35:g,49:c,36:a,37:a,
38:c,39:u,40:c,41:a,42:g,43:a,44:a,45:u,46:c,47:u,48:a,58:g,59:a,
60:a,67:g,71:c,106:g,72:g,105:c,73:g,104:c,74:u,103:g,75:c,102:g,
76:g,101:u,77:c,100:g,78:u,79:g,80:c,97:g,81:c,96:g,82:g,95:c,83:g,
94:c,84:g,93:u,85:c,92:g,86:g,91:c,87:g,88:c,89:a,90:a,98:u,99:g,
107:a,108:a,109:c,120:c]
write_list(1,120),
pair(121,121),
pair(2,119),
pair(0,0),
base(1,a),
pair(17,69),
base(13,a),
base(14,c),
base(15,u),
base(26,a),
base(27,g),
base(28,u),
base(37,a),
base(38,c),
base(39,u),
base(40,c),
base(41,a),
base(42,g),
base(43,a),
base(44,a),
base(45,u),
base(46,c),
base(47,u),
base(48,a),
base(36,c),
bulge(53,54),
base(53,u),
base(54,a),
base(59,a),
base(60,a),
base(25,c),
base(58,g),
bulge(67,67),
base(67,g),
pair(71,106),
base(70,g),
base(16,a),
base(88,g),
base(89,u),
base(87,g),
base(90,a),
base(78,u),
base(79,g),
base(98,u),
base(99,g),
base(107,a),
base(108,a),
base(109,c),
base(12,c),
base(120,c),
s(1,120,[1:a,2:c,119:g,3:g,118:c,4:c,117:g,5:g,116:c|...]),
internal(25,28,58,60),
internal(78,79,98,99) ?
```

Bibliography

[1] S. Abdennadher, T. Frühwirth, and C. H. (Eds.). Special issue on constraint handling rules. *Journal of Theory and Practice of Logic Programming (TPLP)*, 2005.

[2] H. Abramson and V. Dahl. *Logic Grammars.* Springer-Verlag, 1989.

[3] Rosalia Aguirre-Hernandez, Anne Condon, and Holger H. Hoos. Rnasoft: a suite of rna secondary structure prediction and design software tools. *Nucleic Acids Research*, 13(31):3416–3422, 2003.

[4] Tatsuya Akutsu and Satoru Miyano. On the approximation of protein threading. In *RECOMB '97: Proceedings of the first annual international conference on Computational molecular biology*, volume 1, pages 3–8. ACM Press, 1997.

[5] Mirela Andronescu, Anthony P. Fejes, Frank Hutter, Anne Condon, and Holger H. Hoos. A new algorithm for RNA secondary structure design. *Journal of Molecular Biology*, 336(3):607–624, 2004.

[6] P. Baldi. *Bioinformatics: the machine learning approach.* Cambridge, Mass. :MIT Press, 1998.

[7] A. Barranco-Mendoza. Stochastic and heuristic modelling for analysis of the growth of pre-invasive lesions and for a multidisciplinary approach to early cancer diagnosis. *Ph.D. thesis*, 2005.

[8] A. Barranco-Mendoza, D. R. Persaoud, and V. Dahl. A property-based model for lung cancer diagnosis. In *RECOMB '04:8 th Annual Int. Conf. on Computational Molecular Biology.* ACM Press, 2004.

[9] A. Barranco-Mendoza, D. R. Persaoud, and V. Dahl. A property-based model for lung cancer diagnosis. *Int. Conf. on Computational Molecular Biology (RECOMB 2004)*, 2004.

[10] M. Bavarian and V. Dahl. Constraint-based methods for biological sequence analysis. In *PROLE'05*, 2005.

[11] M. Bavarian and V. Dahl. RNA secondary structure design using constraint handling rules. In *Workshop on Constraint Based Methods for Bioinformatics*, 2005.

[12] G. Bes and P. Blache. Proprieties et analyse d'un langage. *TALN99*, 1999.

[13] G. Bes, V. Dahl, D. Guillot, L. Lamadon, I. Milutinovici, and J. Paulo. Parsing system for balanced parenthesis in nl texts. *Poster and Demo at the Lorraine-Saarland Workshop on Prospects and Advances in the Syntax/Semantics Interface*, 2003.

[14] P. Blache and J. M. Balfourier. Property grammars: a flexible constraint-based approach to parsing. *IWPT-2001*, 2001.

[15] Janusz M. Bujnicki. *Practical Bioinformatics.* Springer, 2004.

[16] J. M. Burke and A. Berzal-Herranz. In vitro selection and evolution of RNA: applications for catalytic RNA, molecular recognition and drug discovery. *FASEB J.*, 7:106–112, 1993.

[17] J. J. Cannone, S. Subramanian, M. N. Schnare, J. R. Collett, L. M. D'Souza, Y. Du, B. Feng, N. Lin, L. V. Madabusi, K. M. Muller, N. Pande, Z. Shang, N. Yu, and R.R. Gutell. The comparative RNA web (crw) site: An online database of comparative sequence and structure information for ribosomal, intron, and other RNAs. *BioMed Central Bioinfo.*, 3(2), 2002.

[18] H. Christiansen. CHR as grammar formalism, a first report. *Sixth Annual Workshop of the ERCIM Working Group on Constraints*, 2001.

[19] H. Christiansen. Logical grammars based on constraint handling rules. In *18th International Conference on Logic Programming*, volume 18, page 481. Lecture Notes in Computer Science, 2002.

[20] H. Christiansen. CHR grammars. *International Journal in Theory and Practice of Logic Programming*, 2005.

[21] Peter Clote and Rolf Backofen. *Computational Molecular Biology: An Introduction.* Mathematical and Computational Biology. John Wiley & Sons LTD, 2000.

[22] J. Cohen. Computational molecular biology: A promising application using logic programming and constraint logic programming. *Lecture Notes in Artificial Intelligence*, 1999.

[23] Jacques Cohen. Bioinformatics: an introduction for computer scientists. *ACM Comput. Surv.*, 36(2):122–158, 2004.

[24] V. Dahl and P. Blache. Directly executable constraint based grammars. *Journees Francophones de Programmation en Logique avec Contraintes*, pages 149–166, 2004.

[25] V. Dahl and P. Tarau. Assumptive logic programming. In *ASAI 2004*, 2004.

[26] V. Dahl, P. Tarau, and R. Li. Assumption grammars for processing natural language. *Fourteenth International Conference on Logic Programming*, pages 256–270, 2004.

[27] V. Dahl and K. Voll. Concept formation rules: An executable cognitive model of knowledge construction. In *First International Workshop on Natural Language, Understanding and Cognitive Sciences*, volume 1, pages 28–36. INSTICC Press, 2004.

[28] Jitender S. Deogun, Ruben Donis, Olga Komina, and Fangrui Ma. RNA secondary structure prediction with simple pseudoknots. In *Proceedings of the second conference on asia-pacific bioinformatics*, volume 2, pages 239–246, 2004.

[29] David Eppstein, Zvi Galil, and Raffaele Giancarlo. Speeding up dynamic programming. In *Proc. 29th Symp. Foundations of Computer Science*, volume 29, pages 488–496. IEEE, October 1988.

[30] T. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, 37:95–138, 1998.

[31] T. Frühwirth, A. Di Pierro, and H. Wiklicky. Probabilistic constraint handling rules. *11th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2002)*, 76, 2002.

[32] P. Gueneau de Novoa and K. P. Williams. The tmrna website: reductive evolution of tmrna in plastids and other endosymbionts. *Nucleic Acids Research*, 32:D:104–108, 2004.

[33] Ivo L. Hofacker, Walter Fontana, Peter F Stadler, L Sebastian Bonhoeffer, Manfred Tacker, and Peter Schuster. Fast folding and comparison of RNA secondary structures. *Monatsh. Chem.*, 125:167–188, 1994.

[34] Antony M. Jose. Ribozyme therapy: RNA enzymes to the rescue. *Yale Journal of Biology and Medicine*, 75:215–219, 2002.

[35] R. D. King, S Muggleton, R. A. Lewis, and M. J. E. Sternberg. Drug Design by Machine Learning: The Use of Inductive Logic Programming to Model the Structure-Activity Relationships of Trimethoprim Analogues Binding to Dihydrofolate Reductase. *PNAS*, 89(23):11322–11326, 1992.

[36] Ross D. King. Applying inductive logic programming to predicting gene function. *AI Mag.*, 25(1):57–68, 2004.

[37] P. S. Klosterman, M. Tamura, S. R. Holbrook, and S. E. Berner. Scor: A structural classification of RNA database. *Nucleic Acids Research*, 30(1):392–394, 2002.

[38] R. H. Lathrop and T. F. Smith. Global optimum protein threading with gapped alignment and empirical pair score functions. *Journal of Molecular Biology*, 255:641–665, 1996.

[39] Arthur M. Lesk. *Introduction to Bioinformatics.* Oxford University Press, 2002.

[40] Rune B. Lyngsø, Michael Zuker, Christian N., and S. Pedersen. Fast evaluation of internal loops in RNA secondary structure prediction. *Bioinformatics*, 15(6):440–445, 1999.

[41] Rune B. Lyngsø, Michael Zuker, and Christian N. S. Pedersen. Internal loops in RNA secondary structure prediction. In *RECOMB '99: Proceedings of the third annual international conference on Computational molecular biology*, volume 3, pages 260–267, 1999.

[42] D.H. Mathews, J. Sabina, M. Zuker, and D.H. Turner. Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *J. Mol. Bio.*, 288:911–940, 1999.

[43] David W. Mount. *Bioinformatics: sequence and genome analysis.* Gold Spring Harbor Laboratory Press, 2001.

[44] S. Muggleton. Inductive logic programming. In *ILP-91*, 1991.

[45] S. Muggleton. *Inductive Logic Programming.* Academic Press, London, 1992.

[46] S. Muggleton, R. D. King, and M. J. E. Sternberg. Protein secondary structure prediction using logic-based machine learning. *Protein Engineering*, 5:647–657, 1992.

[47] J. Ngo and J. Marks. Computational complexity of a problem in molecular structure prediction. *Protein Engineering*, 5(4):313–321, 1992.

[48] R. Nussinov and A. Jacobson. Fast algorithm for predicting the secondary structure of single stranded RNA. In *Proc. Natl. Acad. Sci. (USA)*, volume 77, pages 6309–6313, 1980.

[49] R. A. Overbeek. Invited tutorial: Logic programming and genetic sequence analysis. *JICSLP*, 1992.

[50] Y. Sakakibara, M. Brown, R. Hughey, S. Mian, K. Sjolander, R. Underwood, and D. Haussler. Stochastic context-free grammars for trna modeling. *Nucleic Acid Research*, 22:5112–5120, 1994.

[51] T. Schrijvers and T. Frühwirth. CHR website. *www.cs.kuleuven.ac.be/ dtai/projects/CHR/*, May 2005.

[52] Steffen Schulze-Kremer. *Molecular Bioinformatics: Algorithms and Applications.* Walter de Gruyter.Berlin. New York., 1996.

[53] D. B. Searls. *The computational linguistics of biological sequences.* Lecture Notes in Computer Science, 1993.

[54] D. B. Searls. Grand challenges in computational biology. *in Computational methods in molecular biology*, 1998.

[55] Jon Sneyers, Tom Schrijvers, and Bart Demoen. The computational power and complexity of constraint handling rules. In *CHR2005*, 2005.

[56] Maciej Szymanski, Mirsolawa Z. Barciszewska, Volker A. Erdmann, and Jan Barciszewki. 5s ribosomal RNA database. *Nucleic Acids Research*, 30:176–178, 2002.

[57] N. Usman and J. McSwiggen. Catalytic RNA (ribozymes) as drugs. In *Annual Reports in Medicinal Chemistry*, volume 30, pages 285–294. Academic Press Inc., 1995.

[58] K. Voll. *A Methodology of Error Detection: Improving Speech Recognition in Radiology.* PhD thesis, Simon Fraser University, 2006.

[59] H. Wang and D. A. Hickey. Evidence for strong selective constraint acting on the nucleotide composition of 16s ribosomal RNA genes. *Nucleic Acids Research*, 30:2501-2507, 2002.

[60] M. S. Waterman. *Introduction to computational molecular biology: Maps , sequences and genome.* Chapman&Hall, 1995.

[61] E. Winfree, F. Liu, L. Wenzler, and N. Seeman. Design and self-assmbly of 2D DNA crystals. In *Nature*, volume 394, pages 539–544, 1998.

[62] M. Zahariev, V. Dahl, and A. Levesque. Efficient algorithms for the discovery of oligonucleotide signatures for dna sequences and groups of sequences. *(Technical Report)*, 2005.

[63] M. Zuker. On finding all suboptimal foldings of an RNA molecule. *Science*, 244:48–52, April 1989.

[64] M. Zuker. mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Res.*, 31(13):3406–3415, 2003.

[65] M. Zuker. Lectures on RNA secondary structure prediction. *http://www.bioinfo.rpi.edu/ zukerm/lectures/RNAfold-html/*, January 2006.

[66] Michael Zuker and Patrick Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucl.Acids Res.*, 9(1):133–148, 1981.