STRUCTURE-BASED MULTIPLE RNA SEQUENCE ALIGNMENT AND FINDING RNA MOTIFS

Michael Wayne Sarver

A Dissertation

Submitted to the Graduate College of Bowling Green
State University in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2006

Committee:

Craig Zirbel, Advisor

Robert Boughton,
Graduate Faculty Representative

Truc Nguyen

Gabor Szekely

John Chen

ABSTRACT

Craig Zirbel, Advisor

With the advent of faster computers and the availability of RNA crystal structures we can now use more information to align homologous RNA sequences. We can take a crystal structure and construct a probabilistic model, based on a SCFG, of an RNA molecule. We construct objects called nodes that modularize the model into small pieces that are more manageable. Using this model we can take sequences that are similar to the sequence in the 3D crystal structure and look for the most probable way that the model could have generated the sequence. Then we can get a detailed description of how each node of the model could have generated the sequence. Using this information we can align sequences. Given a seed alignment we give a procedure to construct a 3D structural alignment quickly. In addition we show how the parameters from the model can be estimated. We also have the ability to do motif swaps using objects called alternative nodes.

We have developed an algorithm to quickly search through RNA 3D structures to find motifs. This is accomplished by taking a query motif with m bases and finding the center of the heavy atoms for each base and then rotating it onto candidate motifs that have the same number of bases. Then we measure how good a fit the candidate is to the query by using a discrepancy that we define which involves the distance between bases and their relative orientations. A simple inequality allows us to quickly identify candidates whose discrepancy with the query motif will exceed a cutoff discrepancy. We use this to screen out the vast majority quickly.

To my grandmother and grandfather.

ACKNOWLEDGMENTS

First and foremost I want to thank my wife, Amy, for always believing in me and pushing me harder than anyone else. Without her I'm not sure I would have even gone to college. I would like to thank Dr. Zirbel for his infinite patience and all the time he spent helping me with various aspects of writing this dissertation. Asking Dr. Zirbel to be my advisor was one of the smartest decisions I have ever made. I wish to thank the members of my committee, Dr. Chen, Dr. Szekely, Dr. Nguyen, and Dr. Boughton for taking the time to read and evaluate my dissertation. Finally, I would like to thank Nate Iverson for creating the BGSU LaTeX class.

Table of Contents

CHAPTER 1: Introduction **1**

1.1 What is RNA? 1

1.2 Isostericity 3

1.3 What is an RNA multiple sequence alignment? 5

1.4 Why do alignments? 7

1.4.1 Predict secondary structure 7

1.4.2 Infer global 3D structure 7

1.4.3 Phylogeny 8

1.4.4 Predict 3D structure - motif swaps 8

1.4.5 Allows you to check your isostericity conjectures 9

1.5 Alignment procedures 9

1.5.1 Hand alignment 9

1.5.2 Progressive alignment 9

1.5.3 Stochastic Context Free Grammars 10

1.5.4 Covariance Model 11

1.6 Why do we want to do multiple alignments? 13

1.6.1 Why do alignments when there are already alignments available? . . 13

1.6.2 Why do 3D structural alignments? 13

CHAPTER 2: SCFG-MRF model for RNA sequences **14**

2.1 Nodes 15
2.2 Initial Node 18
2.3 Basepair Node 19
2.3.1 Constructing the basepair model 19
2.3.2 Summing over all possible values of (L, R) adds to 1 20
2.4 Junction Node 22
2.5 Hairpin Node 23
2.6 Parsing 23
2.6.1 Parsing an Initial node 25
2.6.2 Parsing a Basepair 26
2.6.3 Parsing a junction 27
2.6.4 Parsing with a Hairpin Node 28
2.7 Cluster nodes and Markov random fields 28
2.7.1 Constructing the Cluster Model 30
2.7.2 Showing the Markov property through an example 32
2.7.3 Generating a cluster 34
2.7.4 Parsing a subsequence using a cluster node 34
2.8 Junction Cluster 35
2.8.1 Constructing Junction Cluster Model 35
2.8.2 Parsing a J_c-node 36
2.9 Alternative 36
2.9.1 Parsing an Alternative node 38
2.9.2 Results 38
2.10 Traceback 40
2.11 Limits on what each node looks at 41

CHAPTER 3: Parameter Estimation **45**

3.1 The Likelihood Function for an Initial Node 45

3.1.1 The MLE for λ_n and ρ_n . . . 46
3.1.2 The MLE for ξ and ζ . . . 48
3.2 The Likelihood Function for a Basepair . . . 49
3.2.1 The MLE for d . . . 50
3.2.2 The MLE for P . . . 50
3.2.3 MLE Using Partitions . . . 52
3.2.4 MLE Using Partitions across different interaction families . . . 53
3.2.5 The MLE for λ_n and ρ_n . . . 55
3.2.6 The MLE's for ξ and ζ . . . 55
3.3 The Likelihood Function for Basepairs in a Cluster Node . . . 56
3.3.1 The MLE for P in a Cluster Node . . . 56
3.4 MLE for a J_c-node . . . 58
3.5 Parameter estimates from data . . . 58

CHAPTER 4: Motif Searching **62**

4.1 Our first approach . . . 63
4.2 Current way of finding pentahedra . . . 64
4.3 Maximum number of candidates . . . 67
4.4 The definition of discrepancy . . . 68
4.5 Permutation of bases . . . 69
4.5.1 Why do we need to permute the bases? . . . 69
4.5.2 How do we permute the bases? . . . 69
4.6 Displaying Candidates . . . 70

CHAPTER 5: Selected Matlab Code **73**

5.1 Matlab code for Cluster Nodes . . . 73
5.2 Matlab code for JunctionCluster Nodes . . . 76
5.3 Matlab code for parsing Alternatives . . . 78

5.4 Matlab code for finding pentahedra 80

5.5 Matlab code for permuting the bases 81

BIBLIOGRAPHY **83**

List of Figures

1.1 23S Haloarcula Marismortui RNA Molecule 1
1.2 5S Haloarcula Marismortui RNA Molecule 2
1.3 5S secondary structure 3
1.4 Basepairs [5] 4
1.5 Interacting Edges and Glycosidic Bond Orientations from LSW paper [5] . . 5
1.6 Unaligned Sequences of archaeal 5S Haloarcula Marismortui 7
1.7 Seed alignment of archaeal 5S Haloarcula Marismortui 7
1.8 Secondary structure example for CM 12
1.9 CM for the secondary structure given in Figure (1.8) 12

2.1 5S Haloarcula Marismortui secondary structure 15
2.2 3D annotation of the 5S RNA Haloarcula Marismortui 16
2.3 Example of a tree structure with nodes 17
2.4 Randomly generated Sequences from an Initial Node 19
2.5 Randomly generated Sequences from a Basepair Node 22
2.6 Junction From 5S Haloarcula Marismortui 23
2.7 Hairpin from 5S Haloarcula Marismortui 24
2.8 Kink-turn from 23S Haloarcula Marismortui 28
2.9 Renumbered Kink-turn from 23S Haloarcula Marismortui 32
2.10 Alternative Tree Structure 37
2.11 Alignment of randomly generated sequences using an alternative node 39

2.12 Alignment of randomly generated sequences using an alternative node 39

2.13 Alignment of randomly generated sequences using an alternative node 40

2.14 Alignment of Archaeal and Bacterial Loop E 41

2.15 Maximum Probability Parse with no restrictions 42

2.16 Maximum Probability Parse with sequence length restriction 43

2.17 Maximum Probability Parse near seed alignment 43

3.1 Alignment of 5S RNA sequences using ad hoc parameters 60

3.2 Alignment of 5S RNA sequences using estimated parameters 61

4.1 Formation of 3 base triples 65

4.2 Creating tetrahedra 66

4.3 Creating pentahedra 67

4.4 Display of candidates using FR3D 71

4.5 Superimposed motifs 72

List of Tables

1.1 Isosteric families for cWW 6
1.2 Isostericity Matrix for G-C cWW basepair 6

2.1 Seed Alignment 44

3.1 Ad hoc Isostericity Matrix for cWW A-U,C-G basepairs 58
3.2 Data Matrix for cWW A-U,C-G basepairs 58
3.3 New Data Matrix for cWW A-U,C-G basepairs 59

4.1 Values of b for spheres of different radii from the 23S ribosomal RNA molecule. 68

CHAPTER 1

Introduction

1.1 What is RNA?

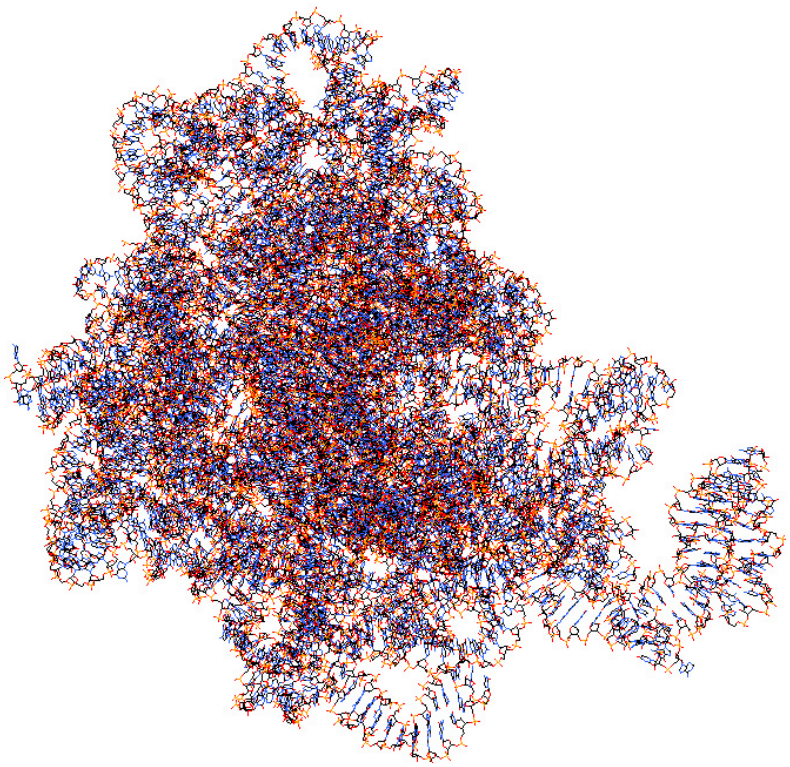

Figure 1.1: 23S RNA Molecule, from 1s72 crystal structure, part of the ribosome, which manufactures proteins. Display created with Swiss-PdbViewer [3]

RNA is a single-stranded nucleic acid molecule involved in protein synthesis and many

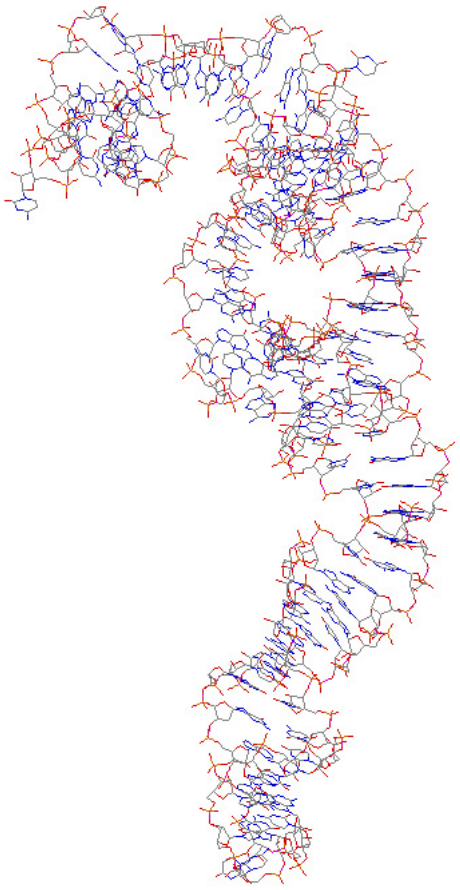

Figure 1.2: 5S RNA Molecule, another part of the ribosome. Display created with Swiss-PdbViewer [3]

other cellular functions. RNA is mostly made up of the four bases adenine, cytosine, guanine, and uracil, denoted by A,C,G, and U respectively. RNA is similar to DNA. The bases A,C, and G that are found in RNA are the same bases found in DNA. In DNA uracil is replaced by thymine. From Figure (1.2) we can see that even though an RNA molecule is single-stranded it can fold back onto itself and form interesting structures.

Protein synthesis by the ribosome depends crucially on the RNA molecule having the specific shape shown here, which is held together by *basepairs*. The most common basepairs are AU and CG (shown in figure (1.2)), but others also occur.

Figure (1.3) gives a 2D representation of what is displayed in Figure(1.2). The figure shows the pattern of basepairs in the molecule. We will call the structure above loop A helix

1, to the right of loop A helix 2, and below loop A helix 3.

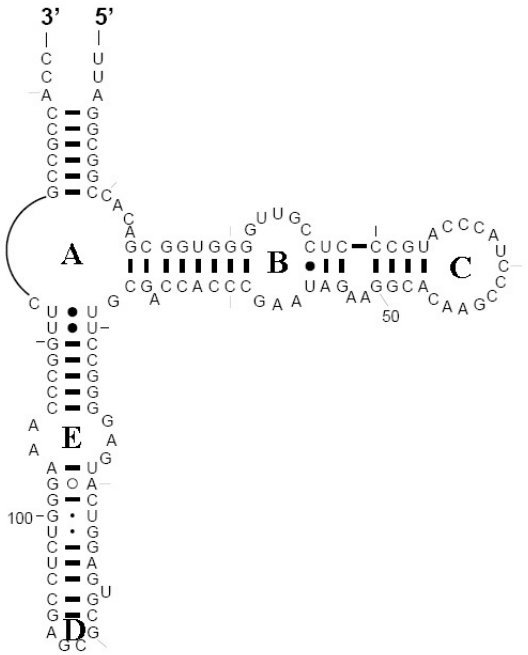

Figure 1.3: Secondary structure of the 5S ribosomal RNA Haloarcula Marismortui

1.2 Isostericity

Over the course of many generations, random mutations will occur in an RNA molecule. Sometimes basepairs are replaced by different basepairs and the shape of the molecule or the function of the molecule or both is unchanged. If the shape remains unchanged then there must be something that the two basepairs have in common. In this case we say that the two basepairs are *isosteric* to one another. That is, one basepair can be replaced by another and it doesn't change the shape of the molecule. If we have two basepairs that have roughly the same C1'-C1' distance and their glycosidic bonds have the same relative orientation then we say that the two basepairs are isosteric.[5] The C1' atoms are labeled as solid black dots in Figure (1.4).

Basepairs can be classified into families. For each base there are three interacting edges. In Figure (1.5) we can see these edges. They are called the Watson-Crick edge, Hoogsteen

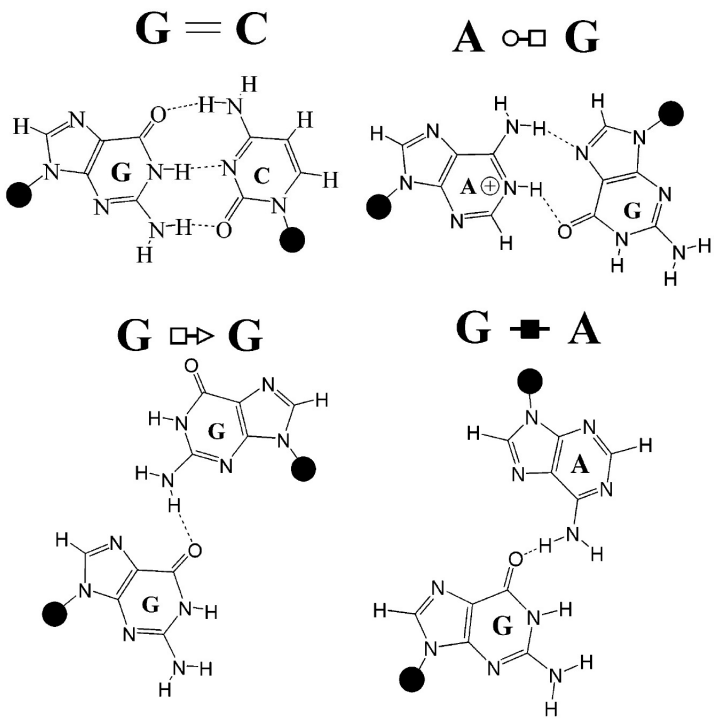

Figure 1.4: Basepairs [5]

edge, and Sugar edge.

Referring to the right hand side of Figure (1.5) we see the orientation of the glycosidic bonds indicated by the arrows. If both arrows are pointing to the same side then we call it a *cis* basepair, otherwise we call it *trans*. Using the hierarchy that the Watson-Crick edges are referred to first followed by Hoogsteen then Sugar, there are twelve different families. Basepairs that belong to the same family (cWW,tWW,cWH,...) generally have glycosidic bonds that have roughly the same orientation. But their C1'-C1' distance can still be different and therefore not all basepairs in a family are isosteric. Therefore, we make *isostericity matrices*. An isostericity matrix is a 4×4 matrix that, for each family, tells which basepairs are isosteric to one another. Below is the table for a cWW interaction.

From Table (1.1) we can see that the canonical basepairs A-U, C-G, G-C, U-A are isosteric to one another because they fall in isostericity family I_1. Also note that the G-G position in the matrix is blank. This is because there is no way for the Watson-Crick edge of guanine nucleotides to form hydrogen bonds, due to electrostatic repulsion.

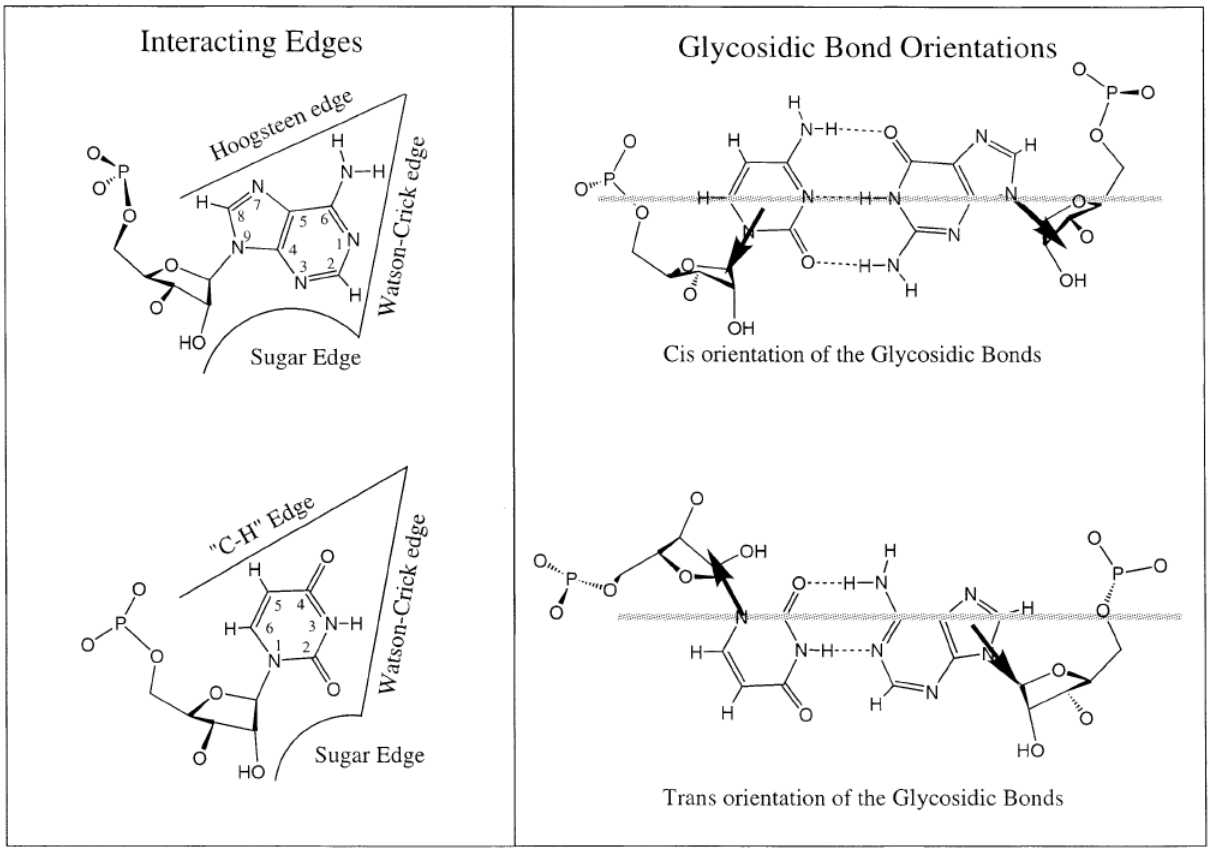

Figure 1.5: Interacting Edges and Glycosidic Bond Orientations from LSW paper [5]

Some basepairs we classify as being *nearly isosteric*. That is, their C1'-C1' distances and relative orientations of their glycosidic bonds are close but not close enough to be considered isosteric. A further subclass are basepairs that are not close enough to be considered nearly isosteric but they are still physically possible. These basepairs are called *allowed*. A basepair that is physically impossible is called *not allowed*. Table 1.2 shows what basepairs are isosteric, nearly isosteric, allowed, and not allowed, indicated by 2, 1, 0, and -3 respectively, for a G-C Cis Watson-Crick basepair.

1.3 What is an RNA multiple sequence alignment?

The 3D structures we have shown and discussed are hard to produce. Much easier is to determine the sequences of bases in a given RNA molecule for a given organism. Different

cWW	A	C	G	U
A	I_4	i_2	I_3	I_1
C	I_2	I_6	I_1	I_5
G	I_3	I_1		i_2
U	I_1	I_5	I_2	I_6

Table 1.1: Isosteric families for cWW

cWW	A	C	G	U
A	0	1	0	2
C	1	0	2	0
G	0	2	-3	1
U	2	0	1	0

Table 1.2: Isostericity Matrix for G-C cWW basepair

molecules may play different roles in different organisms, but molecules which play the same biological role in two organisms and which share similar 3D structures are said to be *homologous*.

When analyzing RNA sequences we would like to know what are some of the common features of the sequences and can we tell anything about the shape of the molecule. An RNA multiple sequence alignment is when we take homologous sequences, RNA molecules that are descended from a common ancestor, and we line up the columns to indicate which bases play the same role in the molecule. In practice, since we don't know the 3D structure of all the sequences, we line them up in a way that either maximizes some *score* or probability.

A typical scoring system is one in which high positive scores will be given to alignments when letters in two columns which we think represent a basepair match a pattern such as A-U, U-A, C-G, or G-C. [7] If a sequence aligns well, has a high score or probability, to a sequence for which we know something about the 3D structure, then we can infer that the other sequence also has a similar 3D structure.

Figure (1.6) shows 10 unaligned sequences. From this figure it is hard to see if the sequences are similar. Note that the sequences have different lengths; some molecules simply have fewer bases than others. In Figure (1.7) we see the same 10 sequences but now they are aligned. Now it is quite obvious that the sequences are very similar.

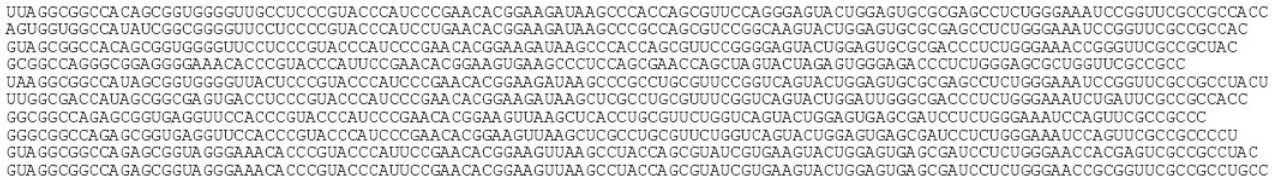

Figure 1.6: Unaligned Sequences of archaeal 5S Haloarcula Marismortui

This alignment, Figure (1.7), is mostly a matter of identifying where the difference in the number of total bases occur. Here it is clear that most of the differences occur at the beginning and the end of the sequences. This corresponds to differences in the length of helix 1 in Figure (1.3).

Figure 1.7: Seed alignment of archaeal 5S Haloarcula Marismortui

1.4 Why do alignments?

1.4.1 Predict secondary structure

The original use of alignments was to use sequence data alone to predict secondary structures, as in Figure (1.3), before any 3D structures were available. There are several techniques to predict secondary structure. These include the Nussinov algorithm and the Zuker folding algorithm [1]

1.4.2 Infer global 3D structure

We know what the 23S looks like since we have a crystal structure. If we have a homologous sequence that aligns well to the 23S then we can infer that the 3D structure of the homologous sequence is very similar to the 3D structure of 23S.

1.4.3 Phylogeny

If two sequences align well then this tells us something about the phylogeny, which is the evolutionary link between the two organisms. There are algorithms available, such as UPGMA and Parsimony, that make phylogenetic trees[1]. These procedures create a measure of distance between two aligned sequences. The smaller the distance, the more closely the two sequences, and thus the organisms, are related. The requirement for these algorithms is a good multiple alignment. Unfortunately, current multiple alignment procedures can only parse helical regions with any accuracy. Therefore when creating a phylogenetic tree non-helical regions have to be ignored since they are not aligned well. Our 3D structure-based multiple alignment procedure can align non-helical as well as helical regions. It is reasonable to assume that if we have a better alignment then we should be able to generate a more accurate phylogenetic tree. We can also define a different distance than the one usually used in Parsimony, based on isosteric substitutions. When isosteric substitutions are made in an RNA molecule the shape and function of the molecule remains unchanged. Therefore, we can set the distance between a pair and one isosteric to it as zero. We could also define distances between pairs that were nearly isosteric or just allowed. All of this will lead to a better understanding of phylogeny.

1.4.4 Predict 3D structure - motif swaps

Some RNA molecules have very similar structures. They just differ in a few locations. For example, the bacterial and archaeal 5S have nearly the same 3D structure. In fact, if we superimpose the two structures we see that they have nearly the same shape. The main difference between the two is the loop E region. If we are given a sequence that is either bacterial or archaeal then by parsing the sequence according to both bacterial and archaeal models, we may be able to determine what type of sequence it is based on how the loop E region was parsed.

Aligning sequences can help us predict 3D structure from sequence data alone. From

the RNA 3D structures we have available one can build a database of models for *motifs*, collections of up to 20 bases. We can then align a sequence or portion of a sequence with an unknown 3D structure using all the models in our database. The alignment that gives the best score would then be our best guess as to what the true 3D structure of the sequence is.

1.4.5 Allows you to check your isostericity conjectures

We can align sequences based on our isostericity rules. If the alignment score is high and there is diversity in the sequences we have reason to believe that our isostericity rules are true. Once we create our alignments we can then re-estimate isostericity substitution parameters, which tell the probabilities for making Isosteric, nearly Isosteric, allowed, and not allowed substitutions. How to estimate isostericity parameters will be discussed in Chapter 3.

1.5 Alignment procedures

1.5.1 Hand alignment

Biologists have created a few multiple alignments by hand either by aligning the sequences from scratch or modifying existing alignments. While these alignments are usually good they take a very long time to produce and therefore it is impractical to do for large sequences or when doing a multiple alignment on a large number of sequences.

1.5.2 Progressive alignment

A progressive alignment is a procedure that constructs a succession of pairwise alignments. First, two sequences are aligned by a pairwise alignment procedure. Then, a third sequence is aligned to the first alignment and so on until all the sequences have been aligned. The problem with this procedure is that it is a heuristic procedure that depends on the order that the sequences were aligned. It is important to align the sequences that are more closely

related first. The upside to this approach is that it is fast and the alignments are usually not too bad. [2]

1.5.3 Stochastic Context Free Grammars

Stochastic Context Free Grammars (SCFG's) are part of Chomsky's hierarchy.[1] They consist of a set of terminals and nonterminals as well as a set of production rules. Terminals are usually denoted with lowercase letters and nonterminals with capital letters. For example, let $\{S, W_1, W_2, W_3\}$ be the set of nonterminals and $\{a, c, g, u\}$ be the set of terminals. Let the production rules be given by

$$\begin{aligned} S &\rightarrow aW_1g|cW_1u \\ W_1 &\rightarrow aW_2u|cW_2g|aW_1g \\ W_2 &\rightarrow aW_2u|cW_2g|cW_3g \\ W_3 &\rightarrow uccg|ggga \end{aligned} \tag{1.1}$$

The symbol | in (1.1) can be read as "or". Then the first production rule says that if we start at S our first production is either aW_1g or cW_1u. Production rule 3 generates au or cg basepairs. Assume that S goes to aW_1g. Then from W_1 we can get either aW_2u or cW_2g or aW_1g. let us pick cW_2g. Then, so far we have

$$S \rightarrow aW_1g \rightarrow acW_2gg \tag{1.2}$$

Now maybe W_2 goes to cW_2g. And then from W_2 to cW_3g. Finally we let W_3 produce $ggga$. Combining all this we get

$$S \rightarrow aW_1g \rightarrow acW_2gg \rightarrow accW_2ggg \rightarrow acccW_3gggg \rightarrow acccgggagggg \tag{1.3}$$

The above is an example of a context free grammar. To make it into a SCFG all we need

to do is to create probability distributions for the production rules. That is, for example, if we take the first production rule in (1.1), we could say that S goes to aW_1g with probability p_{S1} and aW_1g with probability p_{S2}. Likewise we could set the probability that W_1 produces aW_2u, cW_2g, or aW_1g as $p_{w_1 1}$, $p_{w_1 2}$, or $p_{w_1 3}$ respectively. Doing this gives a way to randomly generate sequences. But we could also look at it from the other direction. We could have a sequence that we want to know what was the most probable way the grammar generated the sequence. This is called parsing. SCFGs can be used to model sequence variability for RNA molecules because they can preferentially generate basepairs of a specified type.

1.5.4 Covariance Model

A *covariance model* (CM) is a SCFG that uses an annotated secondary structure, which is obtained from a secondary structure prediction algorithm, to determine the production rules for the grammar. The set of nonterminals are given by the letters S, P, L, R, B, E.[1]

The production rules are similar to the rules given by (1.1) and are defined as

$$\begin{aligned} S &\rightarrow W \\ P &\rightarrow aWb \\ L &\rightarrow aW \\ R &\rightarrow Wa \\ B &\rightarrow SS \\ E &\rightarrow \epsilon \end{aligned} \tag{1.4}$$

Where W$\in \{P, L, R, B, E\}$ and $a, b \in$ {A,C,G,U} and ϵ is the empty string.

The production rules are set in such a way as to mimic the secondary structure. For example, if the secondary structure of a molecule is given by Figure (1.8) we can construct a CM with the form given by Figure (1.9).

Then we can construct a CM that has the structure given by (Figure 1.9) to model the

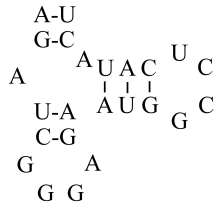

Figure 1.8: Secondary structure example for CM

molecule.

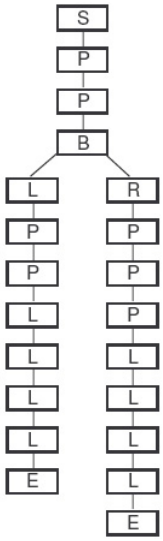

Figure 1.9: CM for the secondary structure given in Figure (1.8)

What we do in chapter 2 is similar to a CM. The difference is that we use a 3D structure to construct the model and therefore we have more information available and we introduce new rewriting rules (cluster, alternative). This allows us to construct a more comprehensive model.

1.6 Why do we want to do multiple alignments?

1.6.1 Why do alignments when there are already alignments available?

While the procedures described do a reasonable job of aligning sequences they have yet to capture the intuition of the trained biologist. Most alignment procedures are based only on sequence data and they use ad hoc parameters. The algorithms are designed to create as many canonical basepairs as possible (helical regions). The problem with aligning sequences this way is that helical regions only account for approximately 70% of the interacting basepairs. By just using sequence data we miss out on 30% of the information. It has been shown that the non-helical regions are not aligned well.[6] In addition, there are problems with the alignment of the helical regions also.[6] Clearly there is room for improvement here. If we use more information we should get better alignments

1.6.2 Why do 3D structural alignments?

A 3D structure of a molecule gives us information that is not available from a set of sequences alone. In addition to showing the locations of all the canonical basepairs the 3D structure shows where all the insertions and non-canonical basepairs are. Using this extra information we can make more elaborate models of RNA molecules. We can model situations where bases are are involved in multiple interactions. Therefore, we can model the 30% of the molecule that the other alignment procedures don't take into account. This will give us a more informed model and therefore a better multiple alignment of the helical as well as the non-helical regions.

CHAPTER 2

SCFG-MRF model for RNA sequences

Our research group has created a program that can analyze an RNA crystal structure and then display a 2D representation that tells what bases are interacting and what type of interaction it is. From this information we can create a model for the molecule. Our working hypothesis is that closely-related organisms have molecules with similar 3D structure, with minor evolutionary variations. More distantly-related molecules may differ more. We wish to make a probabilistic model for sequence generation based on the 3D structure and use it to understand (parse) sequences from other organisms, whether closely or distantly related.

A *secondary structure* is a 2D representation of an RNA molecule that shows where the canonical basepairs are and which gives some idea of the shape of the molecule. Figure (2.1) shows the secondary structure, inferred from multiple sequence alignments based only on canonical basepairs, for the 5S ribosomal RNA of Haloarcula Marismortui. Solid lines indicate canonical basepairs. Non-canonical basepairs occur in the regions marked B,C,and E but are not shown in Figure (2.1). Figure (2.2) shows, in addition to the canonical basepairs, the non-canonical basepairs that are present in the 5S RNA Haloarcula Marismortui. This information is derived from the crystal structure given in Figure (1.2).

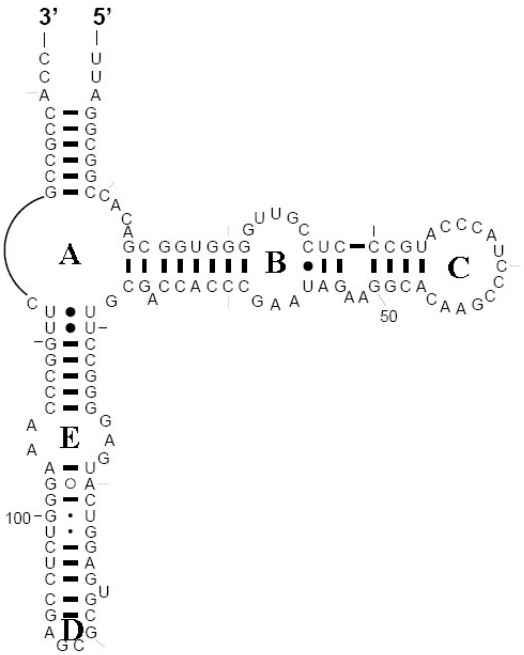

Figure 2.1: Secondary structure of the 5S ribosomal RNA Haloarcula Marismortui, as inferred from sequence alignments.

The location of the 5' is where the RNA starts and it ends at the 3' side. This particular molecule is 122 bases long. If we start where the 5' and 3' are and we move toward loop **A** we can see that the molecule branches and forms other loops/helices.

2.1 Nodes

The model we are about to describe is an SCFG model. The non-terminals are called *nodes* and we organize them into a tree. Also, MRF is explained in Section (2.7). In terms of generation the nodes are capable of generating letters with some dependence structure. The letters that the nodes generate will be independent from node to node. More complicated dependence will be considered at a later date. The nodes that are used are determined directly from the 3D structure, rather than being inferred from a sequence alignment.

There are seven different node types : (I)nitial, (B)asepair, (C)luster, (J)unction, (J_c)unction Cluster, (A)lternative, and (H)airpin. The nodes will be described in detail later.

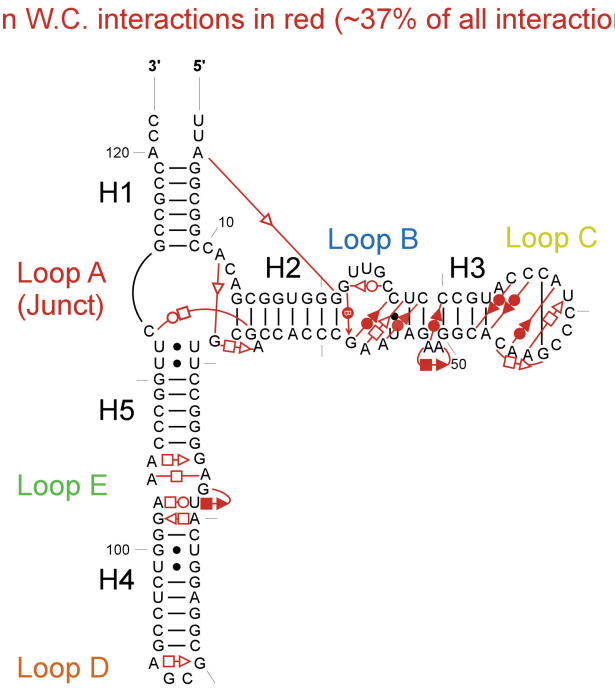

Figure 2.2: 3D annotation of the 5S RNA Haloarcula Marismortui [6]

Using these nodes connected in a tree we can model most situations, but not pseudoknots [8] or other long-range interactions.

To model an RNA molecule we start by creating a tree structure to connect the nodes which generate parts for the sequence. At the root of the tree will be an I-node and an H-node will be at the end of every branch. In Figure (2.1), loops C and D are hairpins. The rest of the tree is constructed in such a way that it generates basepairs and insertions in the desired way.

The *child nodes* for a node n are all the nodes after n such that they are directly connected to node n in the tree structure. We denote these nodes as c_n. Most of the time c_n is one number. However, in some cases there is more than one child node, and then c_n is a vector that contains the numbers of all the child nodes, as we will explain below.

Example: Using nodes to generate a sequence.

Referring to Figure (2.3) we would like to see what kind of sequence that configuration of nodes could generate. First, Node 1 could generate an A on the left and nothing

on the right. Then we can write $N_1 \rightarrow \mathrm{A}N_2$. Node 2 generates a G-C basepair. Therefore, we have $\mathrm{A}N_2 \rightarrow \mathrm{AG}N_3\mathrm{C}$. It is possible that node 3 generates no letters. Then we get $\mathrm{AG}N_3\mathrm{C} \rightarrow \mathrm{AG}N_4\mathrm{C}$. The junction node tells us that we are now going to branch and form other loops. In particular those loops will start at nodes 5 and 9. We can write this as $\mathrm{AG}N_4\mathrm{C} \rightarrow \mathrm{AG}N_5N_9\mathrm{C}$. For nodes 5,6, and 7 assume they generate a U on the right, A-U basepair, and a U-A basepair with an inserted G on the right respectively. This is written as $\mathrm{AG}N_5N_9\mathrm{C} \rightarrow \mathrm{AG}N_6\mathrm{U}N_9\mathrm{C} \rightarrow \mathrm{AGA}N_7\mathrm{UU}N_9\mathrm{C} \rightarrow \mathrm{AGAU}N_8\mathrm{GAUU}N_9\mathrm{C}$. The hairpin node , node 8, now generates an GGGA to close the loop.

$$\mathrm{AGAU}N_8\mathrm{GAUU}N_9\mathrm{C} \rightarrow \mathrm{AGAUGGGAGAUU}N_9\mathrm{C}.$$

Now starting with node 9 we repeat the process we did for the first 8 nodes. So if nodes 9,10,11,12,and 13 generate no letters, A-G basepair, CCA on the left and UG on the right, C-G basepair, and UCCG respectively then, the sequence our model generated is AGAUGGGA-GAUUACCACUCCGGUGGC.

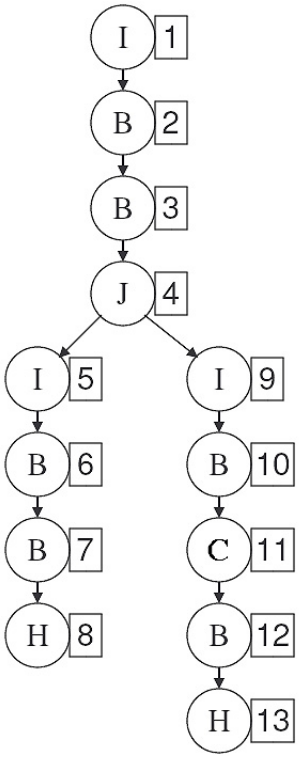

Figure 2.3: Example of a tree structure with nodes

2.2 Initial Node

An *Initial node* (I-node) can generate unpaired bases to the left and right. Let L and R denote the letters generated on the left and right respectively. Then L and R take values in $\Omega = \{e, A, C, G, U, AA, AC, \ldots\}$ where e is a null character. L and R are independent random variables. In the generation phase, an I-node first generates the number of insertions to be made on the left and right according to independent truncated Poisson distributions Λ and $\mathfrak{R}$ with parameters λ and ρ respectively. The distributions are truncated at a user-defined value b, which could be ∞. Therefore,

$$\Lambda(k) = \begin{cases} c_\lambda \frac{\lambda^k e^{-\lambda}}{k!} & 0 \leq k \leq b \\ 0 & \text{otherwise} \end{cases}, \qquad \mathfrak{R}(k) = \begin{cases} c_\rho \frac{\rho^k e^{-\rho}}{k!} & 0 \leq k \leq b \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where c_λ and c_ρ are constants such that $\sum_{k=0}^{b} \Lambda(k) = 1$ and $\sum_{k=0}^{b} \mathfrak{R}(k) = 1$.
The inserted letters on the left and right are generated independently with probability distributions $\xi(\cdot)$ and $\zeta(\cdot)$. Now we can write down the joint distribution of (L, R) which is given by

$$\mathbb{P}((L,R) = (\ell, r)) = \begin{cases} \Lambda(|\ell|)\mathfrak{R}(|r|) \left(\prod_{j=1}^{|\ell|} \xi(\ell(j))\right) \left(\prod_{j=1}^{|r|} \zeta(r(j))\right) & |\ell|, |r| > 0 \\ \Lambda(|\ell|)\mathfrak{R}(0) \left(\prod_{j=1}^{|\ell|} \xi(\ell(j))\right) & |\ell| > 0, |r| = 0 \\ \Lambda(0)\mathfrak{R}(|r|) \left(\prod_{j=1}^{|r|} \zeta(r(j))\right) & |\ell| = 0, |r| > 0 \\ \Lambda(0)\mathfrak{R}(0) & |\ell|, |r| = 0 \end{cases} \tag{2.2}$$

If we set $\prod_{j=1}^{|\ell|} \xi(\ell(j))$ and $\prod_{j=1}^{|r|} \zeta(r(j))$ equal to 1 if $|\ell| = 0$ or $|r| = 0$ respectively then (2.2) can be written as

$$\mathbb{P}((L,R) = (\ell, r)) = \Lambda(|\ell|)\mathfrak{R}(|r|) \left(\prod_{j=1}^{|\ell|} \xi(\ell(j))\right) \left(\prod_{j=1}^{|r|} \zeta(r(j))\right) \tag{2.3}$$

Figure (2.4) shows 10 sequences that were randomly generated using an initial node with $b = 10$, $\xi = [.15, .30, .40, .15]$, $\zeta = [.25, .50, .15, .10]$, $\lambda = 2$, and $r = 3$:

```
UGA...    AA
G  ...   CCC
   ...     C
C  ...  UCUC
AGU...GCCGAA
CC ... CGAAA
CU ... CUCGC
GC ...     C
CA ...CAGAUC
C  ...  GCAU
```

Figure 2.4: Randomly generated Sequences from an Initial Node

2.3 Basepair Node

To simplify the situation we will not take into account basepair stacking interactions. That is, we will assume that a basepair is generated independently of what basepairs are near it. The model should be capable of generating basepairs along with insertions. There should also be a possibility that no letters are generated. i.e. the basepair is deleted. The generation probabilities for a basepair will be guided by isostericity. Therefore each basepair could have its own substitution probabilities.

2.3.1 Constructing the basepair model

A basepair node can generate a pair, a pair with one or more insertions on each side, or nothing. Like I-nodes we let L and R denote the letters generated on the left and right respectively. Then L and R take values in Ω. For example, if L='GA' and R='C' then the node generated a $G - C$ basepair with an inserted A after the G. L and R are dependent random variables that have the property that if either L or R takes the value e then both are equal to e. Hence we have $\mathbb{P}(L = e \text{ and } R \neq e) = \mathbb{P}(L \neq e \text{ and } R = e) = 0$. The

first thing that needs to be determined is whether the node is deleted or not. Let d be the probability that the node is deleted. If the node is deleted then $L = e$ and $R = e$. If the node is not deleted then it generates a basepair. Let $P(\sigma, \alpha\beta)$ be a function that gives the probability of generating the basepair (α, β) given the information σ about the 3D structure. Then $P(\sigma, \cdot)$ can be thought of as a 16 dimensional vector that gives the probability for all 16 possible pairwise combinations of A, C, G, U. Once we have a basepair we then generate a random number of insertions on the left and right according to independent truncated Poisson distributions, just like we did with I-nodes, with parameters λ and ρ. And, just like I-nodes, the inserted letters on the left and right are generated independently with probability distributions $\xi(\cdot)$ and $\zeta(\cdot)$. Now we can write down the joint distribution of (L, R) which is given by

$\mathbb{P}((L, R) = (\ell, r))$

$$= d1_{\{S_n^1\}} + \left[(1-d)1_{\{S_n^2\}}\right] P(\sigma, \ell(1)r(1))\Lambda(|\ell|)\mathfrak{R}(|r|) \left(\prod_{j=2}^{|\ell|} \xi(\ell(j))\right) \left(\prod_{j=2}^{|r|} \zeta(r(j))\right) \quad (2.4)$$

Where S_n^1 is the set of all s, $1 \leq s \leq S$, such that $(\ell_n^s(1), r_n^s(1)) = (e, e)$ and S_n^2 is the set of all s, $1 \leq s \leq S$, such that ($\ell_n^s(1) \neq e$ and $r_n^s(1) \neq e$).

2.3.2 Summing over all possible values of (L, R) adds to 1

We want to show that

$$\mathbf{S} \equiv \sum_{\ell, r} \left(d1_{\{S_n^1\}} + \left[(1-d)1_{\{S_n^2\}}\right] P(\sigma, \ell(1)r(1))\Lambda(|\ell|)\mathfrak{R}(|r|) \left(\prod_{j=2}^{|\ell|} \xi(\ell(j))\right) \left(\prod_{j=2}^{|r|} \zeta(r(j))\right)\right) \quad (2.5)$$

equals 1. We can rewrite this as

$$\mathbf{S} = d + (1-d) \sum_{\ell, r \neq e} \left(P(\sigma, \ell(1) r(1)) \Lambda(|\ell|) \mathfrak{R}(|r|) \left(\prod_{j=2}^{|\ell|} \xi(\ell(j)) \right) \left(\prod_{j=2}^{|r|} \zeta(r(j)) \right) \right)$$

$$= d + (1-d) \left(\sum_{\ell(1), r(1) \neq e} P(\sigma, \ell(1) r(1)) \right) \sum_{\ell' \neq e} \Lambda(|\ell'|) \left(\prod_{j=1}^{|\ell'|} \xi(\ell'(j)) \right) \sum_{r' \neq e} \mathfrak{R}(|r'|) \left(\prod_{j=1}^{|r'|} \zeta(r'(j)) \right)$$

where $\ell'(k) = \ell(k+1)$ and $r'(k) = r(k+1)$ for $k \geq 1$. Then

$$S = d + (1-d) \sum_{\ell' \neq e} \Lambda(|\ell'|) \left(\prod_{j=1}^{|\ell'|} \xi(\ell'(j)) \right) \sum_{r' \neq e} \mathfrak{R}(|r'|) \left(\prod_{j=1}^{|r'|} \zeta(r'(j)) \right)$$

$$= d + (1-d) \left[\sum_{k_1=1}^{b} \Lambda(k_1) \left(\sum_{\substack{\ell' \neq e \\ |\ell'|=k_1}} \left(\prod_{j=1}^{|\ell'|} \xi(\ell'(j)) \right) \right) \right] \left[\sum_{k_2=1}^{b} \mathfrak{R}(k_2) \left(\sum_{\substack{r' \neq e \\ |r'|=k_2}} \left(\prod_{j=1}^{|r'|} \zeta(r'(j)) \right) \right) \right]$$

The probability for the insertions on the left is given by

$$\sum_{\substack{\ell' \neq e \\ |\ell'|=k_1}} \left(\prod_{j=1}^{|\ell'|} \xi(\ell'(j)) \right) = \sum_{\substack{\ell' \neq e \\ |\ell'|=k_1}} \xi(\ell'(1)) \cdots \xi(\ell'(k_1))$$

$$= \left(\sum_{\ell'(1)} \xi(\ell'(1)) \right) \cdots \left(\sum_{\ell'(k_1)} \xi(\ell'(k_1)) \right)$$

$$= 1 \cdots 1 = 1$$

Similarly

$$\sum_{\substack{r' \neq e \\ |r'|=k_2}} \left(\prod_{j=1}^{|r'|} \zeta(r'(j)) \right) = 1$$

therefore we have

$$\mathbf{S} = d + (1-d)\left[\sum_{k_1=1}^{b} \Lambda(k_1)\right]\left[\sum_{k_2=1}^{b} \mathfrak{R}(k_2)\right]$$

By the definition of Λ and $\mathfrak{R}$ in (2.1) we know $\sum_{k_1=1}^{b} \Lambda(k_1) = 1$ and $\sum_{k_2=1}^{b} \mathfrak{R}(k_2) = 1$. Hence,

$$S = d + (1-d) = 1$$

Therefore, (2.4) is a distribution function.

Figure (2.5) shows 10 sequences that were randomly generated using a basepair node with σ =AG Hoogsteen Sugar edge basepair, $b = 10$, $\xi = [.15, .30, .40, .15]$, $\zeta = [.25, .50, .15, .10]$, $\lambda = 3$, $r = 2$, $d = .01$ and $P(\sigma, \cdot) = [0.1343, 0.1343, 0.0009, 0.0182, 0.1343, 0.1343, 0.0009, 0.0009, 0.1343, 0.00$ $0.1343, 0.1343, 0.0009, 0.0009]$

```
CC        ...   A
ACGG      ... CUG
AGUUGG    ...   A
CGG       ...   U
ACGG      ... GAC
ACGA      ... CUG
ACGAGCCC  ...  CG
C         ... UCU
A         ... CGA
UU        ...   G
```

Figure 2.5: Randomly generated Sequences from a Basepair Node

2.4 Junction Node

A junction is where an RNA molecule branches and forms other loops. See Figure (2.6) The simplest kind of junction involves no interactions between these loops. With a junction node there are no letters generated. The node tells where the next two or more nodes are located. So if N_{J_0} is the junction node then it will point to the beginning of all the loops $N_{J_1}N_{J_2}\cdots N_{J_m}$ where $J_0 < J_1 < J_2 < \cdots < J_m$.

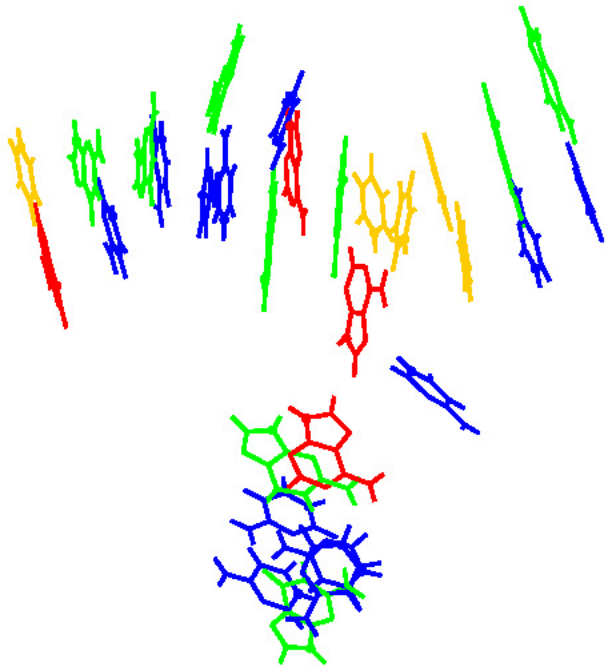

Figure 2.6: Junction From 5S Haloarcula Marismortui

2.5 Hairpin Node

Hairpins are the bases that close a helix. See Figure (2.7). At this point the hairpins generated by the model are very simplistic; one can specify that a hairpin match the pattern GNRA, or that it have 3, 4, or 5 nucleotides, but uniformly distributed within these limits. BGSU student Jesse Stombaugh of our group is currently doing research to better understand hairpins. In the future we will incorporate Stombaugh's findings into the model.

2.6 Parsing

Once we construct a model we would like to take homologous sequences and align them with the model. This is called parsing. If we are parsing a sequence x of length $L = |x|$ and we have N nodes then we create a $N \times L \times L$ matrix mp that stores the maximum probability, for each node n and subsequence that starts at i and ends at j, over all the ways that node n and its children could have generated the subsequence of x that starts at i and ends at j.

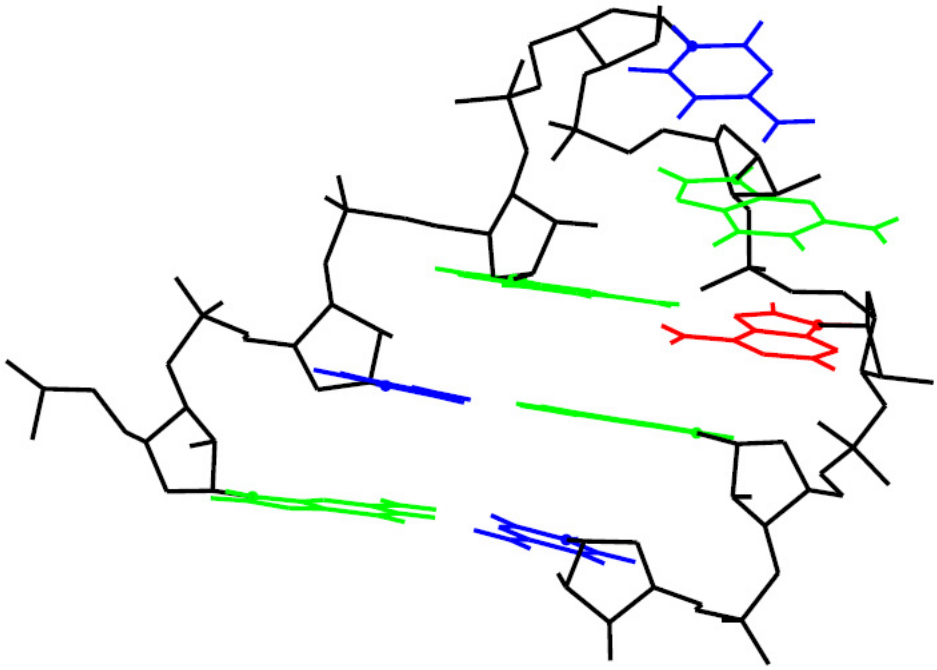

Figure 2.7: Hairpin from 5S Haloarcula Marismortui

For example, $mp(8, 13, 54)$ gives the maximum probability that node 8 and its subsequent children would generate the sequence that started at 13 and ended at 54.

Notation: let $x(i : j)$ be the subsequence of x that starts at i and ends at j.

Parsing is done by dynamic programming. To parse we start by looking at subsequences of length one. For each i, $1 \leq i \leq L$, we start with node $n = N$ and work our way backwards through the nodes. We determine all the possible ways that node n and its subsequent children could have generated subsequence $x(i : j)$ and then we calculate the probability for each possibility. We store the maximum of these probabilities in the matrix mp. So we now know the values of $mp(n, i, i)$ for all n, i. Next we look at subsequences of length two. For each n and $i, i + 1$, $1 \leq i < i + 1 \leq L$, we again determine the different ways in which node n and its children could have generated the sequence $x(i : i + 1)$. But now some of the work has already been done. We are trying to calculate $mp(n, i, i + 1)$. In general there are four possibilities. First, node n could generate both letters in the subsequence. If this is the case

then if node n had children we would have to compute the maximum probability that its children generated no letters. This probability will be zero since every hairpin generates at least one letter, and every loop ends with a hairpin. Second, node n could have generated $x(i)$ and then asked its children how they would generate $x(i+1)$. Third, the roles could be reversed and node n could have generated $x(i+1)$ and then asked its children how they would generate $x(i)$. Finally, node n could generate nothing and ask its child node how it and its children would generate $x(i:i+1)$. This last case illustrates why we have to start from the last node in the tree and work our way back through the nodes. Once we have determined which parse yielded the maximum probability we record how the subsequence was parsed.

In general, to parse the subsequence $x(i:j)$, $1 \leq i \leq j \leq L$, using node n, we have already computed the maximum probability for all contiguous subsequences with length less than $j-i+1$ for all the nodes. We have also computed the maximum probability for all contiguous subsequences with length $j-i+1$ for all nodes k, $k > n$. Therefore we start by determining all the possibilities in which node n and its children could have generated $x(i:j)$. Since we know how c_n parsed all the subsequence, the problem is reduced to looking at all the ways that node n could have generated the first few letters on the left and right and then we ask how node c_n and its children would generate the rest of the subsequence. Again, once we have found the most probable way the model could have generated the subsequence $x(i:j)$ we record how the subsequence was parsed. Once we have parsed all the subsequences we can then use the stored information to trace back through the nodes and determine the most likely way the whole model generated the entire sequence. This procedure will be discussed in Subsection (2.10). This algorithm is attributed to Cocke, Younger, and Kasami. [4]

2.6.1 Parsing an Initial node

Parsing with an I-node is the same as parsing the insertions of a basepair. See below.

2.6.2 Parsing a Basepair

Given a subsequence $x(i:j)$, we wish to determine for each node n, the value of $mp(n,i,j)$. Suppose that node n is a basepair and we want to calculate $mp(n,i,j)$. There are many ways in which node n and its children could have generated $x(i:j)$. First, node n could be deleted so the node looks to see how its child node would generate the subsequence from i to j. Then the probability that node n would generate the subsequence in this way would be given by the deletion probability d_n for node n times the maximum probability that the child of node n would generate the subsequence from i to j. So the probability would be given by

$$\mathbb{P}((L,R)=(e,e)) = d_n \cdot mp(c_n,i,j) \tag{2.6}$$

If node n generated a basepair with no insertions then the probability would be given by $1-d_n$ times the probability that the node generated $(x(i),x(j))$, which is $P(\sigma_n,(x(i),x(j)))$, times the probability for zero insertions on the left and right, $(\Lambda(0)\cdot\mathfrak{R}(0))$, times the maximum probability that the child node generated $x(i+1:j-1)$. i.e.

$$\begin{aligned}&\mathbb{P}(\text{node } n \text{ would generate } x(i:j) \text{ by a basepair } (x(i),x(j)) \text{ with no insertions})\\ &\qquad = (1-d_n)\cdot P(\sigma_n,(x(i)x(j)))\cdot mp(c_n,i+1,j-1)\cdot\Lambda(0)\cdot\mathfrak{R}(0)\end{aligned} \tag{2.7}$$

The basepair node could also generate the sequence as a basepair with an insertion on the left. In this situation we would still have the probability that the basepair was not deleted times the probability for the generation of the basepair $(x(i),x(j))$. Also, the right insertion probability would be the same, $\mathfrak{R}(0)$, but the left insertion probability would be $\Lambda(1)$ times the probability that the inserted base was $x(i+1)$, which is given by $\xi(x(i+1))$. Now, since there are two bases on the left and one on the right the child node would be responsible for the generation of the bases from $i+2$ to $j-1$. Therefore we multiply by the maximum probability that the child node could generate the subsequence from $x(i+2)$ to $x(j-1)$.

Hence,

$$\begin{aligned}&\mathbb{P}(\text{node } n \text{ would generate the basepair } (x(i), x(j)) \text{ with an insertion on the left})\\ &\quad = (1-d_n)\cdot P(\sigma_n,(x(i)x(j)))\cdot mp(c_n,i+2,j-1)\cdot\Lambda(1)\xi(x(i+1))\mathfrak{R}(0)\end{aligned} \tag{2.8}$$

If we consider the possibility of one insertion on the right and none on the left then the probability is given by

$$(1-d_n)\cdot P(\sigma_n,(x(i)x(j)))\cdot mp(c_n,i+1,j-2)\cdot\Lambda(0)\mathfrak{R}(1)\zeta(x(j-1))$$

In general if the basepair node parses the sequence as a basepair with ℓ insertions on the left and r insertions on the right the probability is given by

$$\begin{aligned}\Phi(\ell,r) =&(1-d_n)\cdot P(\sigma_n,(x(i)x(j)))\cdot mp(c_n,i+\ell+1,j-r-1)\\ &\cdot\Lambda(\ell)\left(\prod_{k=1}^{\ell}\xi(x(i+k))\right)\mathfrak{R}(r)\left(\prod_{k=1}^{r}\zeta(x(j-k))\right)\end{aligned} \tag{2.9}$$

Now that we know the probabilities for all the possibilities we are interested in, we choose the configuration that yields the highest probability. Therefore,

$$mp(n,i,j)=\max\left\{d_n\cdot mp(c_n,i,j),\max_{\substack{0\le\ell\le b\\0\le r\le b}}\Phi(\ell,r)\right\}, \tag{2.10}$$

and we also note, for trace back, what the maximizer is.

2.6.3 Parsing a junction

Parsing a junction is easy. If the subsequence we wish to parse is $x(i:j)$ then we need to consider all the locations where the sequence could branch. That is, we want to find the values of $k_1,k_2,\ldots,k_m$ $(i\le k_1<k_2<\cdots<k_m<j)$ that maximizes
$mp(J_1,i,k_1)\cdot mp(J_2,k_1+1,k_2)\cdots mp(J_m,k_m+1,j)$.

2.6.4 Parsing with a Hairpin Node

Since the model for a hairpin is very simple, parsing a hairpin is very easy. A probability of 0 is given to all subsequences of length less than 2 or greater than 5. For GNRA, higher probabilities are given to 4-letters sequences matching GNRA, lower for other 4-letter sequences, 0 for sequences of all other lengths.

2.7 Cluster nodes and Markov random fields

Bases are capable of making more complicated structures than just helices. An RNA molecule can bend, fold back onto itself, bifurcate, or form a number of more complicated interactions. These structures are often called *motifs*. An example of a group of bases forming a complicated structure is a kink-turn, which connects two helices at a sharp angle. See Figure (2.8).

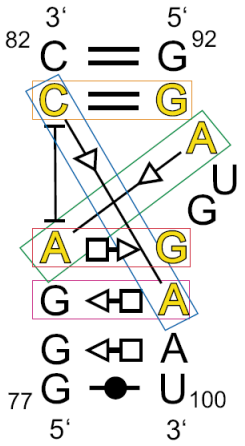

Figure 2.8: Kink-turn from 23S Haloarcula Marismortui, courtesy of J. Stombaugh

The long equals sign indicates that there is a cWW interaction between a C and a G. The circle is for Watson-Crick, square is for Hoogsteen, and the triangle is for Sugar edge. See Figure (1.5). If both bases of a basepair are using the same edge to form the interaction then we just use one circle, square, or triangle and not two with the exception of C-G cWW, represented by double lines, and A-U cWW, represented by a single line (not shown here)[5].

An example of this are bases 77 and 100, which form a cWW interaction.

The bases on the left are GGGACC, on the right GGAUGGAAU reading from the 5' to 3' ends. The first and last are forming a basepair, also the second and second to last, but the next several bases have a more complicated dependence. Presumably isosteric substitutions could be made for some of the pairs. But some nucleotides, such as A80, are involved in more than one basepair.

We need to construct a different tool to generate bases having more complicated dependence. To model these situations, consider a base as a vertex and an interaction between two bases as an edge. At each vertex we have a random variable that can generate an A,C,G, or U. Some of the vertices have dependence between them. If there is an edge between two vertices then the corresponding random variables cannot generate letters independently. Thus, there is long range dependence between random variables that have a path between their vertices. The letters at two vertices are independent if there is no path between their corresponding vertices. Now we break up the graph into the largest subgraphs that are connected graphs. Since we are not modeling stacking interactions, each connected graph will be independent of the other connected graphs.

In the kink-turn, Figure (2.8), there are two connected graphs. Bases 79, 81, 93, and 98 form one graph and 80, 94, and 97 form another. Since the graphs are intertwined, base 80 is between 79 and 81 on the left, we will have to model these two connected graphs together. Now we can define what a cluster is. A *cluster* is a group of connected graphs that are intertwined together with inserted bases. In most cases a cluster will just be a connected graph with inserted bases. Thinking of a cluster this way leads nicely to *Markov Random Fields* (MRF's). Let $X_1, \ldots, X_k$ be random variables taking values in some finite set $\mathbf{S}$, and let $G = (V, E)$ be a finite undirected graph where $V = \{1, \ldots, k\}$. For $v \in V$ let

$$\partial\{v\} = \{a \in V \backslash v | \text{ there exists an edge between } v \text{ and } a\} \tag{2.11}$$

Then, the random variables are said to define a Markov random field if for any $x \in \mathbf{S}^V$,

$$\mathbb{P}(X_i = x_i | X_j = x_j, j \in V \backslash i) = \mathbb{P}(X_i = x_i | X_j = x_j, j \in \partial\{i\}) \tag{2.12}$$

Another way of saying this is that the value at a current location only depends on the valuse at other locations through the values that are near it.[10]

Referring to the kink-turn, Figure (2.8), suppose that we have a sequence that has this same kink-turn. In addition, suppose that we know all the bases in the kink-turn except the base in position 81. We know it is forming a cWW interaction with the base at location 93 and a tSS with the base at location 98. Notice that base 81 also has a long range dependence with the base in position 79. We would like to know the probability that the unknown base is an A,C,G, or U. Clearly the bases at 93 and 98 will play a big role in choice of base that goes into 81. But, what about the base in position 79? If we didn't know the value of the base at 98 then it would have some effect on the value of 81 since it would affect the base at 98. It is conceivable that the way 79 interacts with 98 would rotate 98 so that it can't make certain basepairs with 81. But in this model we deliberately ignore this possibility. Isostericity generally means that glycosidic bond angles remain the same, so the base 98 would have the same orientation. which limits the effect 79 would have on 81.

2.7.1 Constructing the Cluster Model

A C-node can generate a wide range of possibilities. Just like with basepairs we let L and R denote the letters that we generate on the left and right respectively. L and R again take values in Ω. In this case L and R have a more complicated dependence structure than the basepair case. For a basepair there is only one interacting pair, namely $L(1)$ and $R(1)$. With a cluster there are multiple interactions and some bases will interact with more than one base. For example, if $|L| = 4$ and $|R| = 3$ it is possible that the value of $L(4)$ interacts directly with the values of $R(2)$ and $L(1)$. (This will cause problems later when we

estimate parameters.) Again, just like basepairs, we let d be the probability that the node is deleted. If the node is not deleted then it generates letters to the left and right. Let σ be the information about the 3D structure. Then σ tells what kind of graph we have. The joint probability of generating interacting bases is given by

$$\mathbb{P}((L, R, U_\ell, U_r) = (\ell, r, v_\ell, v_r)) = \frac{1}{Z} \prod_{j=1}^{J} P(\sigma(j), \gamma(I_{j1})\gamma(I_{j2})) \tag{2.13}$$

Where Z is chosen for normalization and γ is a list of bases, some from the left and some from the right.

Generation of bases in a C-node is not easy. If there are m bases to be generated then there are 4^m possible realizations. This would be easy to generate if the bases were generated independently. We would just generate m bases each having just 4 possibilities. Since bases can form interactions with more than one base we have to generate all the bases that are dependent at once. So, if there are q bases that are interacting then we have to generate all q bases at once. There are 4^q different possible outcomes for this generation. Generation is easy once we know normalizing constant Z.

C-nodes handle insertions differently than B-nodes. With a B-node the insertions always occur after the interacting basepair. With a C-node there is more than one interacting basepair. Also the insertion might occur in between two different interacting bases and not at the end. Because of this we don't use a truncated Poisson distribution to add insertions. Instead we create *insertion matrices*. An interaction matrix for the left IM_l is a $c_l \times i_l$ matrix where c_l is the number of ways in which the left can form interactions and i_l is the number of bases on the left that are interacting with another base. Each row of IM_l tells which bases on the left are forming interactions and therefore we know which bases are inserted. For example, if

$$IM_l = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 3 & 4 \end{bmatrix} \tag{2.14}$$

then there are two possibilities for the left hand side of the C-node. Either, the first three bases on the left form interactions or the first, third, and fourth are forming interactions and there is an inserted base in the second position. When generating we let U_ℓ and U_r be random variables that determine which row of IM_l and IM_r are to be chosen respectively, with distributions μ_ℓ and μ_r.

Let τ_ℓ and τ_r be the distributions for the inserted letters on the left and right respectively. Let γ be the letters on the left combined with letters on the right. For example if L ='ACG' and R ='CGU' then γ ='ACGCGU'. Let J be the number of interactions and I be a $J \times 3$ matrix that tells which bases are forming interactions and what type of interaction it is. Then the distribution function is given by

$$\begin{aligned}&\mathbb{P}((L,R,U_\ell,U_r)=(\ell,r,v_\ell,v_r))\\&\quad=\frac{1}{Z}\prod_{j=1}^{J}P(\sigma(j),\gamma(I_{j1})\gamma(I_{j2}))\cdot\left[\mu_\ell\prod_{\#\text{ins}}\tau_\ell\right]\cdot\left[\mu_r\prod_{\#\text{ins}}\tau_r\right]\end{aligned}\tag{2.15}$$

2.7.2 Showing the Markov property through an example

Referring to Figure (2.8), renumber the interacting basepairs in the kink-turn. Starting at 79 and going clockwise to 98 renumber the bases as 1 to 7 respectively.

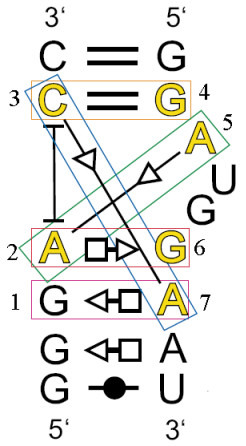

Figure 2.9: Renumbered Kink-turn from 23S Haloarcula Marismortui

There are 5 interaction in this cluster. Let,

$$I = \begin{bmatrix} 1 & 7 & \sigma(1) \\ 2 & 5 & \sigma(2) \\ 2 & 6 & \sigma(3) \\ 3 & 4 & \sigma(4) \\ 3 & 7 & \sigma(5) \end{bmatrix} \tag{2.16}$$

which indicates which bases are interacting.

$$\mathbb{P}((L,R) = (\ell, r)) = \frac{1}{Z} \prod_{j=1}^{J} P(\sigma(j), \gamma(I_{j1})\gamma(I_{j2})) \tag{2.17}$$

We want to calculate

$$\mathbb{P}(X_3 = x_3 | X_j = x_j, j \neq 3) \tag{2.18}$$

Using the definition of conditional probability and the distribution function given by (2.15) we get

$$= \frac{\frac{1}{Z} P(\sigma(1), x_1x_7) \cdot P(\sigma(2), x_2x_5) \cdot P(\sigma(3), x_2x_6) \cdot P(\sigma(4), x_3x_4) \cdot P(\sigma(5), x_3x_7)}{\sum_{n=1}^{4} \frac{1}{Z} P(\sigma(1), x_1x_7) \cdot P(\sigma(2), x_2x_5) \cdot P(\sigma(3), x_2x_6) \cdot P(\sigma(4), \alpha_n x_4) \cdot P(\sigma(5), \alpha_n x_7)} \tag{2.19}$$

If we pull out the terms that don't contain x_3 then,

$$= \frac{\frac{1}{Z} P(\sigma(1), x_1x_7) \cdot P(\sigma(2), x_2x_5) \cdot P(\sigma(3), x_2x_6) \cdot P(\sigma(4), x_3x_4) \cdot P(\sigma(5), x_3x_7)}{\frac{1}{Z} P(\sigma(1), x_1x_7) \cdot P(\sigma(2), x_2x_5) \cdot P(\sigma(3), x_2x_6) \cdot \sum_{n=1}^{4} \left(P(\sigma(4), \alpha_n x_4) \cdot P(\sigma(5), \alpha_n x_7)\right)} \tag{2.20}$$

Therefore,

$$= \frac{\frac{1}{Z} P(\sigma(4), x_3x_4) \cdot P(\sigma(5), x_3x_7)}{\frac{1}{Z} \sum_{n=1}^{4} \left(P(\sigma(4), \alpha_n x_4) \cdot P(\sigma(5), \alpha_n x_7)\right)} \tag{2.21}$$

Rewriting gives us

$$= \mathbb{P}(X_3 = x_3 | X_4 = x_4, X_7 = x_7) = \mathbb{P}(X_3 = x_3 | X_j = x_j, j \in \partial\{3\}) \tag{2.22}$$

Hence we have the Markov property.

2.7.3 Generating a cluster

In order to generate a cluster we first have to find the normalizing constant Z in (2.15). To do this we set

$$Z = \sum_{\ell,r} \prod_{j=1}^{J} P(\sigma(j), \gamma(I_{j1})\gamma(I_{j2})) \tag{2.23}$$

During the normalization process we record all the values of $\prod_{j=1}^{J} P(\sigma(j), \gamma(I_{j1})\gamma(I_{j2}))$ for all possible values of ℓ and r. Next we enumerate the possibilities $t_1, \ldots t_{4^m}$ and we generate a number z between zero and one according to a uniform distribution. Finally, we find the value of j such that $\sum_{i=1}^{j-1} \mathbb{P}(t_i) < z \leq \sum_{i=1}^{j} \mathbb{P}(t_i)$. This gives us the bases that are interacting. Next we have to determine if there are any insertions. Therefore we generate U_ℓ and U_r and so $IM_\ell(U_\ell, \cdot)$ and $IM_r(U_r, \cdot)$ tell where the interacting bases are located and hence where the insertions are. Then, for each insertion on the left we determine which letter to use according to τ_ℓ. For the right we use τ_r.

2.7.4 Parsing a subsequence using a cluster node

Generating a cluster is more difficult than parsing one. If we are given a subsequence $x(i:j)$ that we want to parse we first determine the probability that the node would have generated no letters. This is the deletion probability for the node times the probability that the next node and its children generated the sequence $x(i:j)$. This probability is given by

$$\mathbb{P}((L, R) = (e, e)) = d_n \cdot mp(c_n, i, j) \tag{2.24}$$

Next we calculate the maximum probability the node and its children generated the sequence if the node isn't deleted. To accomplish this we loop through the left and right

insertion possibilities and determine the probabilities for each. Next we loop through the interactions.At each stage we compute

$$\Phi(\ell, r, v_\ell, v_r) = (1 - d_n)\mathbb{P}[(L, R, U_\ell, U_r) = (\ell, r, v_\ell, v_r)] \cdot mp(c_n, a, b) \tag{2.25}$$

Once all the possibilities for Equation (2.28) have been computed we then compute

$$mp(n, i, j) = \max\left\{ d_n \cdot mp(c_n, i, j), \max_{\ell, r, v_\ell, v_r} \Phi(\ell, r, v_\ell, v_r) \right\} \tag{2.26}$$

We also record the values that that gave us the maximum so we can use them later to do the traceback. Refer to Section (5.1) for Matlab code for parsing a cluster node.

2.8 Junction Cluster

A *Junction Cluster* node (J_c-node) is a combination of a J-node and a C-node. We have the same situation as a J-node in that the RNA molecule branches and forms other loops, but now we consider interaction between these loops. And like a C-node, these interactions can be modeled with a Markov Random Field. In addition to interaction between the left and right we now have to consider interaction with bases in the center.

2.8.1 Constructing Junction Cluster Model

We again let L and R denote the letters on the left and right and let C represent the letters in the center. Like L and R, C takes values in Ω. The generation of a J_c-node is done almost the same way that a C-node is. The only difference is that J_c-nodes have letters in the center and are therefore usually larger. The probability density of the J_c-node is almost identical to that of the Cluster. See Equation (2.15). If we define U_c, μ_c, and τ_c for the center as we did for the left and right as in Subsection (2.7.1), and γ as the combinations of the letter on the left, center, and right then, the density for the J_c-node can we written as

$$\begin{aligned}
&\mathbb{P}((L,C,R,U_\ell,U_c,U_r)=(\ell,c,r,v_\ell,v_c,v_r))\\
&=\frac{1}{Z}\prod_{j=1}^{J}P(\sigma(j),\gamma(I_j^1)\gamma(I_j^2))\cdot\left[\mu_\ell\prod_{\#\text{ins}}\tau_\ell\right]\cdot\left[\mu_c\prod_{\#\text{ins}}\tau_c\right]\cdot\left[\mu_r\prod_{\#\text{ins}}\tau_r\right]
\end{aligned}\tag{2.27}$$

2.8.2 Parsing a J_c-node

Parsing a J_c-node combines what is required to parse a C-node and a J-node. Like a C-Node, we have to loop through the left, and right interaction possibilities. In addition we also need to loop through the interaction possibilities for the center which takes into account where the junction occurs. Similar to the C-node case, we evaluate

$$\begin{aligned}
&\Phi(\ell,c,r,v_\ell,v_c,v_r,m)\\
&=\mathbb{P}[(L,C,R,U_\ell,U_c,U_r)=(\ell,c,r,v_\ell,v_c,v_r)]\cdot mp(c_n(1),a,m)\cdot mp(c_n(2),d,b)
\end{aligned}\tag{2.28}$$

And again we find

$$mp(n,i,j)=\max_m\left\{\max_{\ell,c,r,v_\ell,v_c,v_r}\Phi(\ell,c,r,v_\ell,v_c,v_r,m)\right\}\tag{2.29}$$

and recode the arguments that resulted in the maximum. The matlab code for parsing a J_c-node can be found in Section (5.2).

2.9 Alternative

When modeling a section of an RNA molecule we might be unsure what model is the most appropriate so we make several plausible models. We would like to know which model is the best. One way to do this would be to rotate all the sub-models we have into the large model of the molecule and parse each separately and see which one gives the highest score. Unfortunately this could take a long time, especially if the overall model is large or we are

parsing a large number of sequences. A better way to accomplish this task is to use *alternative nodes* (A-node). In the large model of the RNA molecule we put all the sub-models inside the large model. The A-node points to each of the sub-models and it picks the sub-model that has the highest probability of generating the subsequence we are interested in. At the generation stage, the A-node selects randomly between two or more generation mechanisms with probability distribution $\eta(\cdot)$. So, if there are m different alternatives we let $k_1, \ldots, k_m$ be the starting location of the alternatives. Then the probability that the alternative node would pick node k_v is given by $\eta(v)$.

We write the alternatives one after another. Therefore, the first alternative will contain the nodes $k_1, k_1 + 1, k_1 + 2, \cdots, k_2 - 1$. And the second alternative will contain the nodes $k_2, k_2 + 1, k_2 + 2, \cdots, k_3 - 1$. If the last alternative contains the nodes $k_m, k_m + 1, k_m + 2, \cdots, k_{m+1} - 1$ then the nodes $k_2 - 1, k_3 - 1, \ldots, k_{m+1} - 1$ will all point to k_{m+1}. See Figure (2.10).

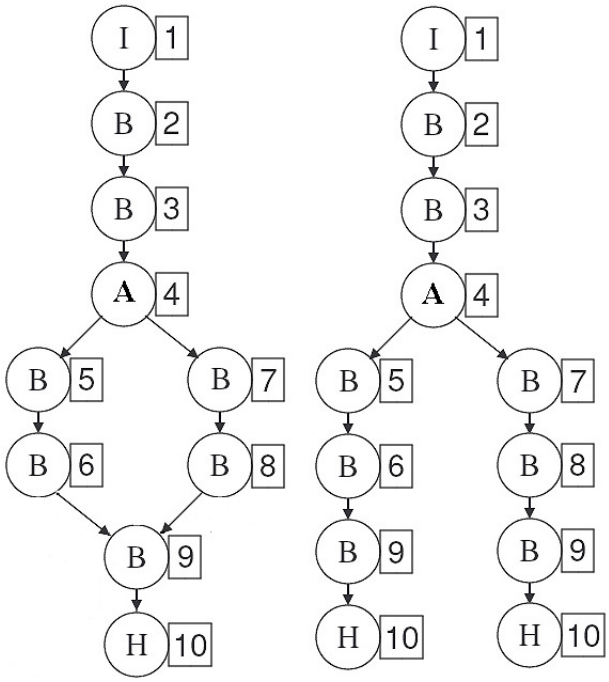

Figure 2.10: The figure on the left shows the structure of a model with an alternative node. The figure on the right shows how a model with an alternative node can be displayed as a tree structure.

2.9.1 Parsing an Alternative node

Assume the subsequence we want to parse starts at i and ends at j and that all the nodes after node k_0 have been parsed. Then to parse the alternative node at k_0 we choose the alternative α such that α is the alternative that maximizes $\eta_{k_0}(a)mp(k_a, i, j)$, $1 \leq a \leq m$. Therefore

$$\alpha = \underset{1 \leq a \leq m}{argmax} \left[\eta_{k_0}(a)mp(k_a, i, j)\right] \tag{2.30}$$

and

$$mp(k_a, i, j) = \eta_{k_0}(\alpha)mp(k_\alpha, i, j) \tag{2.31}$$

2.9.2 Results

Figure (2.11) shows an alignment of sequences that were generated using an alterative node with three alternatives, affecting columns 17 to 34. Node 16 was the alternative node. The models are identical except at six basepair nodes. We used an alternative node to try and determine which sequence came from which model. The first model generated the first five sequences and nodes 17 to 22 were canonical cWW basepairs. The second model generated the next five sequences and its alternative nodes were six AG tHS basepairs, in nodes 23 to 28. The third model generated the last five sequences and its alternative nodes were six AA tWH, nodes 29 to 35. From node 36 onward, the models are the same. The parser was able to correctly determine which alternative generated the sequence. The pluses in the figure mark alternatives that were not chosen.

Figure (2.12) shows essentially the same setup as in Figure (2.11) except that the alternatives are have more commonalities. The first five sequences of Figure (2.12) were generated the same way that the first five sequences of Figure (2.11) were generated. The second five sequences contain an alternative that consists of four nodes that generate from an AG tHS and two nodes that generate basepairs isosteric to canonical basepairs. The third alternative consists of four nodes that generate basepairs isosteric to tWH and two canonical basepair

Figure 2.11: Alignment of randomly generated sequences using an alternative node

nodes. Nodes 1 to 15 and 35 to 49 are canonical cWW and node 50 is a GNRA hairpin. From (2.12) we can see that the parser was able to choose the correct alternative most of the time. Since the alternatives are small and very similar to one another, we would expect that some sequences would be mislabeled.

Figure 2.12: Alignment of randomly generated sequences using an alternative node

A slightly easier case than the one laid out in Figure (2.12) is when the alternatives are of different length. In Figure (2.13) we display 15 sequences where the first 5 sequences were generated with an alternative that contained 5 cWW basepairs. The second 5 consisted of 4 tWH and 2 cWW basepairs. And the last 5 were generated from 5 tWH and 2 cWW. The parser in this case was able to correctly determine which sequence was generated from which alternative.

Figure (2.14) shows the alignment of archaeal and bacterial 5S Loop E using an alternative node. The first 20 sequences are archaea and sequences 21 through 40 are bacterial. With our model the parser was able to determine which sequences were archaeal and which were

Figure 2.13: Alignment of randomly generated sequences using an alternative node

bacterial.

2.10 Traceback

After a sequence has been parsed we need to work our way back through the nodes to see what nodes are responsible for generating which part of the sequence. When parsing with each node we stored information about how the node parsed. Starting with node one we ask it how it parsed the sequence. Node one tells us how many bases to the left and right it would have generated. So if node one would have generated two bases on the left and one on the right we then move to node two and ask it how it would have parsed the sequence starting at three and going to $L_1 - 1$.

When tracing back through a cluster node we again need to know what bases would be generated on the left and the right, but we also need to know which bases were not part of the connected graph and therefore are bulged bases.

When tracing back with a junction we find the location where the junction node said the sequence split and then we ask its children how they would generate the two subsequences.

With alternative nodes we choose the alternative that yielded the highest probability of being generated and label that entire alternative as active and all the other alternatives as inactive. When we go to display the alignment we will know which alternative to display.

When we get to a hairpin we note that it must generate whatever portion of the sequence

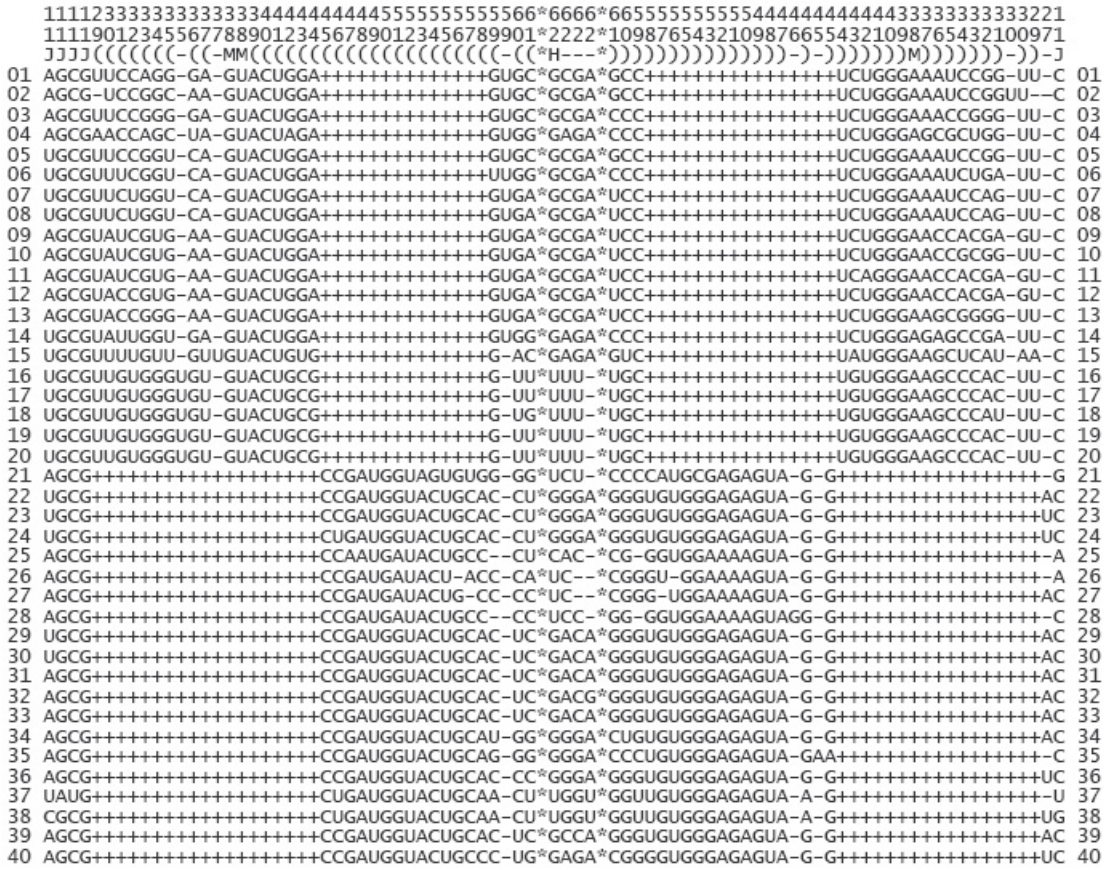

Figure 2.14: Alignment of Archaeal and Bacterial Loop E

that remains, and the we stop the traceback for that chain.

2.11 Limits on what each node looks at

By going through all subsequences i, j $(1 \leq i < j \leq L)$ and nodes $n = 1, \ldots, N$, we are guaranteed to find the maximum probability parse, but the operation count is of order L^3 and therefore for long sequences this could take a long time to compute. Figure (2.15) illustrates the large number of calculations that are done for node 26. Since the number of nodes is proportional to the length of the sequence and $1 \leq i, j \leq L$ the complexity of the algorithm is $\mathrm{O}(L^3)$.

One way to speed up this process is to restrict the length of subsequence that a node looks at. For example, by how the nodes are defined it is impossible for a hairpin node to

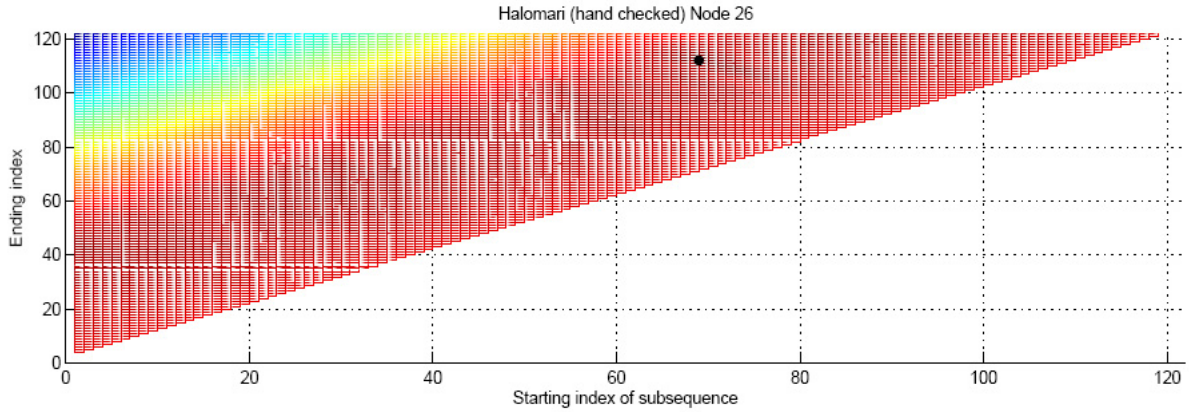

Figure 2.15: Shows the probability that node 26 and its children generated the sequence $x(i:j)$ for $1 \leq i, j \leq L$. The black dot indicates the location of the highest probability.

generate no bases or more than h bases. So it does not make sense to ask how a hairpin node would generate a subsequence with length greater than h since this probability would be zero. For a basepair the situation is similar but a little more complicated. A B-node always has nodes after it that are responsible for generating some part of the sequence. Therefore we don't just put an upper bound on the length of the subsequence that the B-node could generate, we also restrict the minimum length of the subsequence. By doing this though we are not guaranteed to find the maximum probability parse. However, if we are generous in the minimum and maximum length of the subsequence to be parsed for each node, with a high degree of certainty, we will obtain the same maximum probability parse that we get from looking at all contiguous subsequences. The advantage of parsing with this restriction is that the process is now of order L^2. See Figure (2.16). The number of calculations is roughly L times the maxlength-minlength which is constant and therefore doesn't depend on L.

But we can do even better than this. If we begin with a reasonably good alignment and seek only to improve it by 3D structure information we can restrict the values of i, j we consider for each node. Since the input alignment is close to the optimal alignment we only need to adjust the position of the bases slightly when producing the optimal alignment from a seed alignment. By shifting each base to the left or right, if needed, by a few positions we can

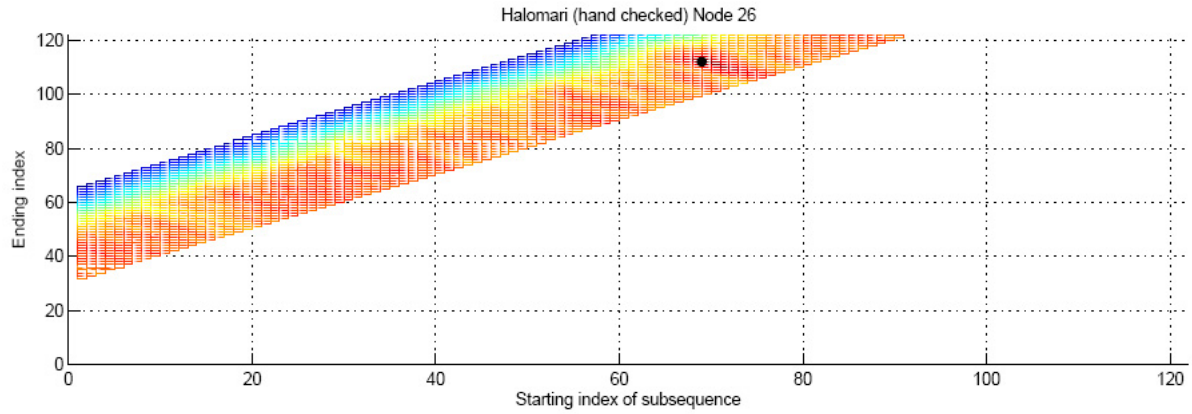

Figure 2.16: Maximum Probability when restricting the minimum and maximum sequence length.

obtain the same optimal alignment that we get when we consider all possible values of i, j. Since we made the model based on the 3D structure we know which nodes are responsible for generating each part of the sequence. Therefore we can restrict which values of i and j we consider. From figure (2.17) we can see that the vast majority of the subsequences are not considered. And since the size of the box does not depend on the length of the sequence to be parsed, the operation count is of order L.

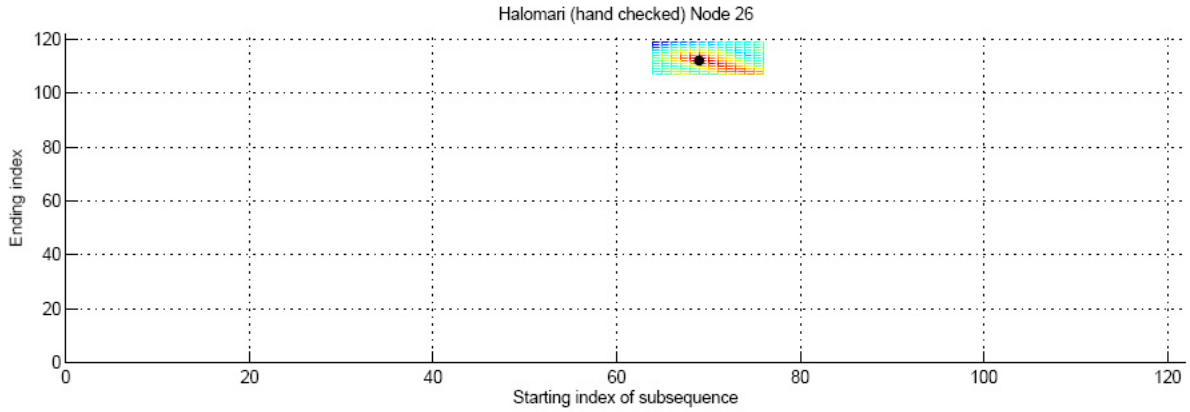

Figure 2.17: Maximum probability when restricting to the bases near the seed alignment

In the seed alignment let the first sequence be the sequence from the 3D structure. In the definition of the nodes we specify which bases from the 3D structure each node is responsible for. Let S_k be the k^{th} sequence without gaps. So, S_k is a string comprised of the letters

A,C,G, and U. Next, let S_k^g be the k^{th} sequence from the seed alignment. Then S_k^g is the same as S_k except that it contains gaps. From there we find a mapping ψ from the index in S_1 to the column number in S_1^g. Next we let ϕ be the identity map from the column number of S_1^g to the column number of S_k^g for all k. Finally we create a function θ_k from the column number of S_k^g to the index number of S_k where if $S_k^g(a)$ is a letter then $\theta_k(a)$ is the position of the letter $S_k^g(a)$ in S_k. If $S_k^g(a)$ is not a letter then let b be the closest number to a such that $S_k^g(a)$ is a letter. If there are two possible values of b then we will choose the smallest. Then $\pi_k = \theta_k \circ \phi_k \circ \psi$ is a mapping from indices of S_1 to indices of S_k. For example, if we are given the seed alignment in Table (2.1) then $\pi_2(\{1,2,3,4,5,6\}) = \{1,2,4,5,6,6\}$

S_1^g	=	A	C	-	-	G	A	-	C	G
S_2^g	=	A	G	-	C	U	A	-	-	C

Table 2.1: Seed Alignment

Since the seed alignments aren't perfect and also there is some variability in the sequences we set a variable δ that restricts how many bases to the left and right of the node to consider. This variable limits the values of i and j that we use to parse each sequence. Then instead of looking at all the possible values of i, j we use our function π and δ to create a list of i, j values that we will consider. For each node we already know what bases they would generate so we take those values and apply π to them. For example, if Node 10 is likely to be responsible for the bases 14,15 and 110 in S_1 then node 10 is responsible for bases $\pi_k(14), \pi_k(15)$, and $\pi_k(110)$ in S_k. If we set $\delta = 10$ then the i values that node 5 will consider for sequence k are $\pi_k(14) - 10$ to $\pi_k(15) + 10$ and the j values will be $\pi_k(110) - 10$ to $\pi_k(110) + 10$. Even with a small δ we can obtain the same parse as we did when we considered all possible values of i, j. This procedure dramatically decreases the time required to parse a sequence since the process is only of order L.

CHAPTER 3

Parameter Estimation

The model we have constructed has many parameters, insertion length, base distribution, base substitution probabilities, interaction probabilities in cluster nodes, hairpin type. If we had sequences generated from the model, still aligned so we knew what nodes generated what letters, we could estimate parameters, that is what we presume here.

3.1 The Likelihood Function for an Initial Node

Fix an initial node n. We take a random sample of size S from our model which includes S sequences, with full alignment information. Let ℓ_n^s and r_n^s be the letters for sequence s that were generated at node n on the left and right respectively. Then the likelihood function is given by

$$L(\theta|x) = \prod_{s=1}^{S} \Lambda_n(|\ell_n^s|)\mathfrak{R}_n(|r_n^s|) \left(\prod_{j=1}^{|\ell_n^s|} \xi(\ell_n^s(j)) \right) \left(\prod_{j=1}^{|r_n^s|} \zeta(r_n^s(j)) \right) \tag{3.1}$$

and the log likelihood is given by

$$\log(L(\theta|x)) = \sum_{s=1}^{S} \left[\log(\Lambda_n(|\ell_n^s|)) + \log(\mathfrak{R}_n(|r_n^s|)) + \sum_{j=1}^{|\ell_n^s|} \log(\xi(\ell_n^s(j))) + \sum_{j=1}^{|r_n^s|} \log(\zeta(r_n^s(j))) \right] \tag{3.2}$$

3.1.1 The MLE for λ_n and ρ_n

Recall that the normalizing constant for Λ_n is:

$$c_{\lambda_n} = \left(\sum_{k=0}^{b} \frac{\lambda_n^k e^{-\lambda_n}}{k!} \right)^{-1} \tag{3.3}$$

Then $c_{\lambda_n} = (f(\lambda_n))^{-1}$ where $f(\lambda_n) = \sum_{k=0}^{b} \frac{\lambda_n^k e^{-\lambda_n}}{k!}$ and its derivative with respect to λ_n is given by

$$\frac{\partial c_{\lambda_n}}{\partial \lambda_n} = -1(f(\lambda_n))^{-2} f'(\lambda_n) = -1(c_{\lambda_n})^2 f'(\lambda_n). \tag{3.4}$$

If we take the derivative of $f(\lambda_n)$ with respect to λ_n we get

$$\begin{aligned} \frac{\partial}{\partial \lambda_n} f(\lambda_n) &= \sum_{k=0}^{b} \frac{k\lambda_n^{k-1} e^{-\lambda_n} - \lambda_n^k e^{-\lambda_n}}{k!} = \sum_{k=1}^{b} \frac{\lambda_n^{k-1} e^{-\lambda_n}}{(k-1)!} - \sum_{k=0}^{b} \frac{\lambda_n^k e^{-\lambda_n}}{k!} \\ &= \sum_{k=0}^{b-1} \frac{\lambda_n^k e^{-\lambda_n}}{k!} - \sum_{k=0}^{b} \frac{\lambda_n^k e^{-\lambda_n}}{k!} = c_{\lambda_n} - \frac{\lambda_n^b e^{-\lambda_n}}{b!} - c_{\lambda_n} = -\frac{\lambda_n^b e^{-\lambda_n}}{b!} \end{aligned} \tag{3.5}$$

We can now find the MLE for λ_n by taking the derivative of the log likelihood function and setting it to zero.

$$\frac{\partial}{\partial \lambda_n} (\log(L(\theta|x))) = \frac{\partial}{\partial \lambda_n} \sum_{s=1}^{S} \log(\Lambda_n(|\ell_n^s|))$$

$$= \frac{\partial}{\partial \lambda_n} \sum_{s=1}^{S} \left[\log(c_{\lambda_n}) + |\ell_n^s| \log(\lambda_n) - \lambda_n - \log(|\ell_n^s|!) \right]$$

$$= \sum_{s=1}^{S} \left[\frac{-c_{\lambda_n}^2 \frac{\lambda_n^b e^{-\lambda_n}}{b!}}{c_{\lambda_n}} + \frac{|\ell_n^s|}{\lambda_n} - 1 \right] = -c_{\lambda_n} S \frac{\lambda_n^b e^{-\lambda_n}}{b!} + \frac{\sum_{s=1}^{S} |\ell_n^s|}{\lambda_n} - S = 0 \tag{3.6}$$

This is a transcendental equation. However, if the cutoff b is much larger than λ_n then

$$-c_{\lambda_n} S \frac{\lambda_n^b e^{-\lambda_n}}{b!} \approx 0 \tag{3.7}$$

So λ_n is approximately the solution of

$$\frac{\sum_{s=1}^{S} |\ell_n^s|}{\lambda_n} - S = 0 \tag{3.8}$$

solving for λ_n we get

$$\hat{\lambda}_n = \frac{\sum_{s=1}^{S} |\ell_n^s|}{S} \tag{3.9}$$

Which is the average number of insertions on the left. As a practical matter, this means we should choose b much larger than the average number of insertions. The value $b = 10$ is sufficient for most purposes. One should note that the MLE in this case might not be unique and that further analysis on the MLE need to be done. Now we need to verify that this is indeed a maximum. Therefore we take the second derivative.

$$\begin{aligned} \frac{\partial^2}{\partial \lambda_n^2}(\log(L(\theta|x))) &= -\frac{S}{b!}\left[c_{\lambda_n}\left(-\lambda_n^b e^{-\lambda_n} + b\lambda_n^{b-1}e^{-\lambda_n}\right) + c_{\lambda_n}^2 \lambda_n^b e^{-\lambda_n}\lambda_n^b e^{-\lambda_n}/b!\right] \\ &= -\frac{S}{b!}c_{\lambda_n}\lambda_n^{b-1}e^{-\lambda}\left[-\lambda_n + b + c_{\lambda_n}\lambda_n^{b+1}e^{-\lambda}/b!\right] + \sum_{s=1}^{S}\left[\frac{-|\ell_n^s|}{\lambda_n^2}\right] \end{aligned} \tag{3.10}$$

If we have $\lambda_n < b$ then $\dfrac{\partial^2}{\partial \lambda_n^2}(\log(L(\theta|x))) < 0$. Therefore we have a maximum. Similarly for ρ_n we get

$$\hat{\rho}_n = \frac{\sum_{s=1}^{S} |r_n^s|}{S} \tag{3.11}$$

Which is the average number of insertions on the right.

3.1.2 The MLE for ξ and ζ

Since we use the same insertion parameters for I and B nodes we let $IB = \{n | n \text{ is a I-node or an B-node}\}$. Then,

$$\frac{\partial}{\partial \xi(\alpha)}(\log(L(\theta|x))) = \sum_{s=1}^{S} \sum_{n \in IB} \sum_{j=1}^{|\ell_n^s|} \left[\frac{1}{\xi(\ell_n^s(j))} 1_{\{\ell_n^s(j)=\alpha\}} \right] = \sum_{s=1}^{S} \sum_{n \in IB} \sum_{j|\ell_n^s(j)=\alpha} \frac{1}{\xi(\alpha)} \tag{3.12}$$

$$= \frac{1}{\xi(\alpha)} \sum_{s=1}^{S} \sum_{n \in IB} \sum_{j|\ell_n^s(j)=\alpha} 1 = \frac{1}{\xi(\alpha)} M_\alpha^\xi \text{ where } M_\alpha^\xi = \sum_{s=1}^{S} \sum_{n \in IB} \sum_{j|\ell_n^s(j)=\alpha} 1 \tag{3.13}$$

Since we want to maximize $\log(L(\theta|x))$ with respect to ξ under the constraint $\sum_{\gamma \in \{A,C,G,U\}} \xi(\gamma) = 1$ we can use Lagrange multipliers. Let $f = \log(L(\theta|x)))$ and $g = \sum_{\gamma \in \{A,C,G,U\}} \xi(\gamma) - 1$. We have already determined that $\frac{\partial f}{\partial \xi(\alpha)} = \frac{1}{\xi(\alpha)} M_\alpha^\xi$. Taking the partial derivative with respect to $\xi(\alpha)$ of g yields

$$\frac{\partial g}{\partial \xi(\alpha)} = 1 \tag{3.14}$$

from (3.13) and (3.14) we get

$$\frac{1}{\xi(\alpha)} M_\alpha^\xi = \lambda \tag{3.15}$$

for all α. Therefore,

$$\frac{1}{\xi(\alpha)} M_\alpha^\xi = \lambda = \frac{1}{\xi(\gamma)} M_\gamma^\xi \tag{3.16}$$

Solving for $\xi(\gamma)$ we get

$$\xi(\gamma) = \xi(\alpha) \frac{M_\gamma^\xi}{M_\alpha^\xi} \tag{3.17}$$

Then, summing over $\gamma \in \{A, C, G, U\}$ we get

$$1 = \sum_{\gamma \in \{A,C,G,U\}} \xi(\gamma) = \sum_{\gamma \in \{A,C,G,U\}} \xi(\alpha) \frac{M_\gamma^\xi}{M_\alpha^\xi} = \xi(\alpha) \frac{\sum_{\gamma \in \{A,C,G,U\}} M_\gamma^\xi}{M_\alpha^\xi} \tag{3.18}$$

Therefore our estimate for ξ is

$$\hat{\xi}(\alpha) = \frac{M_\alpha^\xi}{M^\xi} \tag{3.19}$$

where $M^\xi = \sum_{\gamma \in \{A,C,G,U\}} M_\gamma^\xi$. The estimate $\hat{\xi}(\alpha)$ is the the ratio of times we observe a base of type α on the left over the total number of bases that were inserted on the left.

A similar argument gives us

$$\hat{\zeta}(\alpha) = \frac{M_\alpha^\zeta}{M^\zeta} \tag{3.20}$$

3.2 The Likelihood Function for a Basepair

Fix a basepair n. We take a random sample of size S from our model. Recall that S_n^1 is the set of all s, $1 \leq s \leq S$, such that $(\ell_n^s(1), r_n^s(1)) = (e, e)$) and S_n^2 is the set of all s, $1 \leq s \leq S$, such that ($\ell_n^s(1) \neq e$ and $r_n^s(1) \neq e$). There are no other possibilities, so $S_n^1 \bigcup S_n^2 = \{1, 2, \ldots, S\}$

The likelihood function for node n is given by

$$L(\theta|x) = \prod_{s=1}^{S} \Bigg[d_n 1_{\{s \in S_n^1\}} + \left[(1-d_n)1_{\{s \in S_n^2\}}\right] P(\sigma_n, \ell_n^s(1) r_n^s(1)) \Lambda(|\ell_n^s| - 1) \mathfrak{R}(|r_n^s| - 1) \left(\prod_{j=2}^{|\ell_n^s|} \xi(\ell_n^s(j)) \right) \left(\prod_{j=2}^{|r_n^s|} \zeta(r_n^s(j)) \right) \Bigg] \tag{3.21}$$

Since $S_n^1 \bigcap S_n^2 = \emptyset$, (3.21) can be written as

$$L(\theta|x) = \prod_{s \in S_n^1} d_n \cdot \prod_{s \in S_n^2} \left[(1-d_n)\right] P(\sigma_n, \ell_n^s(1) r_n^s(1)) \Lambda(|\ell_n^s| - 1) \mathfrak{R}(|r_n^s| - 1) \cdot \left(\prod_{j=2}^{|\ell_n^s|} \xi(\ell_n^s(j)) \right) \left(\prod_{j=2}^{|r_n^s|} \zeta(r_n^s(j)) \right) \tag{3.22}$$

Then the log likelihood is given by

$$\log(L(\theta|x)) = \sum_{s\in S_n^1}\log(d_n) + \sum_{s\in S_n^2}\Bigg[\log(1-d_n) + \log(P(\sigma_n, \ell_n^s(1)r_n^s(1))) + \log(\Lambda(|\ell_n^s|-1)) + \log(\Re(|r_n^s|-1)) + \sum_{j=2}^{|\ell_n^s|}\log(\xi(\ell_n^s(j))) + \sum_{j=2}^{|r^s|}\log(\zeta(r^s(j)))\Bigg] \tag{3.23}$$

3.2.1 The MLE for d

To find the MLE for the deletion parameter d_n we first need to take the partial derivative of the log likelihood function, equation (3.23), with respect to d_n and set it equal to zero.

$$\frac{\partial}{\partial d_n}(\log(L(\theta|x))) = \sum_{s\in S_n^1}\frac{1}{d_n} + \sum_{s\in S_n^2}\frac{-1}{1-d_n} = \frac{|S_n^1|}{d_n} + \frac{-|S_n^2|}{1-d_n} = 0 \tag{3.24}$$

Then solving for d_n we get

$$\hat{d_n} = \frac{|S_n^1|}{|S_n^1|+|S_n^2|} = \frac{|S_n^1|}{S}, \tag{3.25}$$

which is the fraction of sequences for which node n is deleted. To verify that (3.25) corresponds to a maximum we look at the second derivative. The second derivative is given by

$$\frac{\partial^2}{\partial d_n^2}(\log(L(\theta|x))) = \sum_{s\in S_n^1}\frac{-1}{d_n^2} + \sum_{s\in S_n^2}\frac{-1}{(1-d_n)^2} \tag{3.26}$$

Since equation (3.26) is always less than zero we indeed have a maximum.

3.2.2 The MLE for P

Recall that $P(\sigma,\cdot)$ is a 16-element vector of substitution probabilities corresponding to an observed basepair of type σ in the 3D structure. $P(\sigma, AU)$ for example is the probability that we have an AU baspair of type σ. All of these probabilities need to be estimated. If the number of different observed basepair types σ is large, there may not be enough data to

estimate all the parameters. We my use the same value of σ for several nodes. Some of the 16 numbers in $P(\sigma, \cdot)$ may also be equal.

Let $U = \{z \in \mathbb{R}^{16} | z \geq 0, \sum_{i=1}^{16} z_i = 1\}$ and fix a basepair type σ^*. We wish to find the value of $P(\sigma^*, \cdot) \in U$ that maximizes $L(\theta|x)$. We start by taking the partial derivative of $\log(L(\theta|x))$ with respect to $P(\sigma^*, \alpha\beta)$.

$$\begin{aligned} \frac{\partial}{\partial P(\sigma^*, \alpha\beta)}(\log(L(\theta|x))) &= \sum_{\substack{s \in S_n^2 \\ n:\sigma_n = \sigma^*}} \frac{1}{P(\sigma_n, \ell_n^S(1) r_n^s(1))} \cdot 1_{\{(\ell_n^s(1) r_n^s(1)) = (\alpha\beta)\}} \\ &= \sum_{\substack{(\ell_n^s(1) r_n^s(1)) = (\alpha\beta) \\ n:\sigma_n = \sigma}} \frac{1}{P(\sigma^*, \alpha\beta)} = \frac{1}{P(\sigma^*, \alpha\beta)} \cdot M_{\alpha\beta}^{\sigma^*} \end{aligned} \tag{3.27}$$

Where $M_{\alpha\beta}^{\sigma^*} = \sum_{\substack{(\ell_n^s(1) r_n^s(1)) = (\alpha\beta) \\ n:\sigma_n = \sigma^*}} 1$, which is the number of times we observe $(\alpha\beta)$ in σ^*-type basepairs, no matter where they occur in the structure. Now, using Lagrange multipliers with $f = \log(L(\theta|x))$ and $g = \sum_{(\gamma\delta)} P(\sigma^*, \gamma\delta) - 1$ the equation $Df = \lambda Dg$, using equation (3.27) and the fact that $\frac{\partial g}{\partial P(\sigma^*, \alpha\beta)} = 1$, becomes $\frac{1}{P(\sigma^*, \alpha\beta)} \cdot M_{\alpha\beta}^{\sigma^*} = \lambda$ for all $\alpha\beta$. Therefore.

$$\frac{1}{P(\sigma^*, \alpha\beta)} \cdot M_{\alpha\beta}^{\sigma^*} = \lambda = \frac{1}{P(\sigma^*, \gamma\delta)} \cdot M_{\gamma,\delta}^{\sigma^*} \tag{3.28}$$

then, solving for $P(\sigma^*, \gamma\delta)$ we get

$$P(\sigma^*, \gamma\delta) = P(\sigma^*, \alpha\beta) \cdot \frac{M_{\gamma,\delta}^{\sigma^*}}{M_{\alpha\beta}^{\sigma^*}} \tag{3.29}$$

Since $\sum_{(\gamma,\delta)} P(\sigma^*, \gamma\delta) = 1$ we have

$$\sum_{(\gamma,\delta)} P(\sigma^*, \gamma\delta) = \sum_{(\gamma,\delta)} P(\sigma^*, \alpha\beta) \cdot \frac{M_{\gamma,\delta}^{\sigma^*}}{M_{\alpha\beta}^{\sigma^*}} = 1 \tag{3.30}$$

therefore

$$\frac{\sum_{(\gamma,\delta)} M_{\gamma,\delta}^{\sigma^*}}{M_{\alpha\beta}^{\sigma^*}} P(\sigma^*, \alpha\beta) = 1 \tag{3.31}$$

and our estimate for P is

$$\hat{P}(\sigma^*, \alpha\beta) = \frac{M^{\sigma^*}_{\alpha\beta}}{\sum_{(\gamma,\delta)} M^{\sigma^*}_{\gamma,\delta}} = \frac{M^{\sigma^*}_{\alpha\beta}}{M^{\sigma^*}} \tag{3.32}$$

where $M^{\sigma^*} = \sum_{(\gamma,\delta)} M^{\sigma^*}_{\gamma,\delta}$. Here $\frac{M^{\sigma^*}_{\alpha\beta}}{M^{\sigma^*}}$ is the number of times that pair $\alpha\beta$ is observed where a σ^* basepair is present, divided by the number of basepairs of the type σ^* that are not deleted.

3.2.3 MLE Using Partitions

Now suppose that some of the 16 parameters in $P(\sigma^*, \alpha\beta)$ are the same. Let $I_1, I_2, \ldots, I_k$ be a partition of the 16 possible basepair types. Then $P(\sigma^*, \alpha\beta)$ has the same value for all $\alpha\beta \in I_i$. We will write $P(\sigma^*, I_i)$ for this common probability. Then the MLE for $P(\sigma^*, I_i)$ can be obtained as follows.

$$\begin{aligned} \frac{\partial}{\partial P(\sigma^*, I_i)}(\log(L(\theta|x))) &= \sum_{\substack{s \in S^2_n \\ n:\sigma_n = \sigma^*}} \frac{1}{P(\sigma_n, \ell^s_n(1) r^s_n(1))} \cdot 1_{\{(\ell^s_n(1) r^s_n(1)) \in I_i\}} \\ &= \sum_{(\ell^s_n(1) r^s_n(1)) \in I_i} \frac{1}{P(\sigma^*, I_i)} = \frac{1}{P(\sigma^*, I_i)} \sum_{(\ell^s_n(1) r^s_n(1)) \in I_i} 1 = \frac{1}{P(\sigma^*, I_i)} \cdot M^{\sigma^*}_{I_i} \end{aligned} \tag{3.33}$$

Where $M^{\sigma^*}_{I_i} = \sum_{(\ell^s_n(1) r^s_n(1)) \in I_i} 1$ is the number of times we observe a pair in set I_i. As above we use Lagrange multipliers with $f = \log(L(\theta|x))$ and $g = \sum_{\ell=1}^{k} |I_\ell| P(\sigma^*, I_\ell) - 1$ to obtain

$$\frac{1}{P(\sigma^*, I_i)} \cdot M^{\sigma^*}_{I_i} = |I_i| \lambda \tag{3.34}$$

and therefore

$$\frac{1}{P(\sigma^*, I_i)} \cdot \frac{M^{\sigma^*}_{I_i}}{|I_i|} = \frac{1}{P(\sigma^*, I_j)} \cdot \frac{M^{\sigma^*}_{I_j}}{|I_j|} \text{ for } 1 \le j \le k \tag{3.35}$$

Solving for $|I_j|P(\sigma_n, I_j)$ and summing over the possible basepair values we get

$$1 = \sum_{j=1}^{k} |I_j|P(\sigma^*, I_j) = \sum_{j=1}^{k} |I_i|P(\sigma^*, I_i) \cdot \frac{M_{I_j}^{\sigma^*}}{M_{I_i}^{\sigma^*}} \tag{3.36}$$

Hence,

$$\hat{P}(\sigma^*, I_i) = \frac{M_{I_i}^{\sigma^*}}{\sum_{j=1}^{k} |I_i| M_{I_j}^{\sigma^*}} = \frac{M_{I_i}^{\sigma^*}}{|I_i| M^{\sigma^*}}, \tag{3.37}$$

which is the fraction of the number of times pair $\alpha\beta$, where $\alpha\beta \in I_i$, is observed where a σ^* basepair is, over the number of basepairs of the type σ^* that are not deleted times the number of elements in I_i. This spreads the probability equally among the elements of I_i.

3.2.4 MLE Using Partitions across different interaction families

Finally, we can imagine that some of the parameters might be the same between different values of σ^*. For example, we can partition the 16 into isosteric families, and use 4 parameters for the basepairs that are isosteric, nearly isosteric, allowed, and not allowed. For each type σ_i^*, let I_g^i, $q = 1, 2, \ldots, k$, partition the set $\{AA, Ac, \ldots, UU\}$

Assume that

$$P(\sigma_1^*, I_q^1) = P(\sigma_2^*, I_q^2) = \cdots = P(\sigma_v^*, I_q^v) \tag{3.38}$$

for $1 \leq q \leq k$. Once again using Lagrange multipliers with $f = \log(L(\theta|x)))$ and $g = \sum_{d=1}^{v} \left[\sum_{\ell=1}^{k} |I_\ell^d| P(\sigma_d^*, I_\ell^d) - 1\right]$, let $\pi_m^d = P(\sigma_d^*, I_m^d)$. Then, since $\pi_m^1 = \pi_m^d$ for all d,

$$\frac{\partial f}{\partial \pi_m^1} = \sum_{d=1}^{v} \sum_{(\ell_n^s(1) r_n^s(1)) \in I_m^d} \frac{1}{\pi_m^d} = \sum_{d=1}^{v} \frac{1}{\pi_m^d} \sum_{(\ell_n^s(1) r_n^s(1)) \in I_m^d} 1$$

$$= \sum_{d=1}^{v} \frac{1}{\pi_m^d} M_{I_m^d}^{\sigma_d^*} \tag{3.39}$$

Now turning our attention to g we obtain

$$\frac{\partial g}{\partial \pi_m^1} = \sum_{d=1}^{v} \sum_{\gamma\delta} 1_{\{\gamma\delta \in I_m^d\}} = \sum_{d=1}^{v} |I_m^d| = I_m. \tag{3.40}$$

where $I_m = \sum_{d=1}^{v} |I_m^d|$. Combining (3.39) and the result from (3.40) we get $\sum_{d=1}^{v} \frac{1}{\pi_m^d \cdot I_m} M_{I_m^d}^{\sigma_d^*} = \lambda$. Therefore,

$$\sum_{d=1}^{v} \frac{1}{\pi_m^d \cdot I_m} M_{I_m^d}^{\sigma_d^*} = \lambda = \sum_{d=1}^{v} \frac{1}{\pi_n^d \cdot I_n} M_{I_n^d}^{\sigma_d^*} \tag{3.41}$$

By our assumption in (3.38) equation (3.41) can be written as

$$\frac{1}{\pi_m^1} \sum_{d=1}^{v} \cdot I_n M_{I_m^d}^{\sigma_d^*} = \lambda = \frac{1}{\pi_n^t} \sum_{d=1}^{v} \cdot I_m M_{I_n^d}^{\sigma_d^*} \tag{3.42}$$

solving for π_n^t and multiplying both sides by $|I_n^t|$we have

$$|I_n^t|\pi_n^t = |I_n^t|\pi_m^1 \frac{I_m \sum_{d=1}^{v} M_{I_n^d}^{\sigma_d^*}}{I_n \sum_{d=1}^{v} M_{I_m^d}^{\sigma_d^*}} \tag{3.43}$$

Summing over the values of n and t

$$v = \sum_{t=1}^{v} \sum_{n=1}^{k} |I_n^t|\pi_n^t = \sum_{t=1}^{v} \sum_{n=1}^{k} |I_n^t|\pi_m^1 \frac{I_m \sum_{d=1}^{v} M_{I_n^d}^{\sigma_d^*}}{I_n \sum_{d=1}^{v} M_{I_m^d}^{\sigma_d^*}} \tag{3.44}$$

Solving for π_m^1 we get

$$\hat{\pi}_m^1 = \frac{v \sum_{d=1}^{v} M_{I_m^d}^{\sigma_d^*}}{I_m \sum_{t=1}^{v} \sum_{n=1}^{k} \left(\frac{|I_n^t|}{I_n} \sum_{d=1}^{v} M_{I_n^d}^{\sigma_d^*} \right)}$$

$$= \frac{v \sum_{d=1}^{v} M_{I_m^d}^{\sigma_d^*}}{I_m \sum_{n=1}^{k} \left(\frac{\sum_{t=1}^{v} |I_n^t|}{I_n} \sum_{d=1}^{v} M_{I_n^d}^{\sigma_d^*} \right)} = \frac{v \sum_{d=1}^{v} M_{I_m^d}^{\sigma_d^*}}{I_m \sum_{n=1}^{k} \sum_{d=1}^{v} M_{I_n^d}^{\sigma_d^*}} \tag{3.45}$$

Notice that if there is only one σ^* (i.e., $v = 1$) then (3.45) reduces to (3.37) and if $v = 1$ and $|I_m^1| = 1$ then (3.45) reduces to (3.32).

3.2.5 The MLE for λ_n and ρ_n

If we let $\ell_n^{'s}(k) = \ell_n^s(k+1)$ for $k \geq 1$ then the likelihood function (3.23) with just the information about insertions can be written as

$$\log(L(\theta|x)) = \sum_{s=1}^{S} \log(\Lambda(|\ell_n^{'s}|)) + \log(\mathfrak{R}(|r_n^{'s}|)) + \sum_{j=1}^{|\ell_n^{'s}|} \log(\xi(\ell_n^{'s}(j))) + \sum_{j=1}^{|r_n^{'s}|} \log(\zeta(r_n^{'s}(j))) \quad (3.46)$$

This has the same form as (3.2). So from equation (3.9) we get

$$\hat{\lambda}_n \approx \frac{\sum_{s \in S_n^2} |\ell_n^{'s}|}{|S_n^2|} \quad (3.47)$$

therefore,

$$\hat{\lambda}_n \approx \frac{\sum_{s \in S_n^2} (|\ell_n^s| - 1)}{|S_n^2|} \quad (3.48)$$

A similar argument gives us

$$\hat{\rho}_n = \frac{\sum_{s \in S_n^2} (|r_n^s| - 1)}{|S_n^2|} \quad (3.49)$$

3.2.6 The MLE's for ξ and ζ

The MLE's for ξ and ζ

The MLE's for ξ and ζ are the same as those obtained in (3.19) and (3.20).

3.3 The Likelihood Function for Basepairs in a Cluster Node

Then the likelihood function is given by

$$L(\theta|x) = \prod_{s=1}^{S} \frac{1}{Z} \prod_{j=1}^{J} P(\sigma_n(j), \gamma_n^s(I_j^1)\gamma_n^s(I_j^2)) \cdot u_n^s \prod_{\#\text{ins}} \tau \cdot v_n^s \prod_{\#\text{ins}} \omega \tag{3.50}$$

where $Z = \sum_a \prod_{j=1}^{J} P(\sigma_n(j), a(I_j^1)a(I_j^2))$ and a is the combination of the left and right letters. Then the log likelihood is given by

$$\begin{aligned} \log(L(\theta|x)) = &\sum_{s=1}^{S}\sum_{j=1}^{J} \log\left(P(\sigma_n(j), (\gamma_n^s(I_j^1), \gamma_n^s(I_j^2)))\right) - S\log(Z) \\ &+ \log(u_n^s) + \sum_{\#ins} \log(\tau) + \log(v_n^s) + \sum_{\#ins} \log(\omega) \end{aligned} \tag{3.51}$$

3.3.1 The MLE for P in a Cluster Node

Now we take the partial derivative with respect to $P(\sigma^*, \alpha\beta)$ and we get

$$\begin{aligned} \frac{\partial}{\partial P(\sigma^*, \alpha\beta)} \log(L(\theta|x)) &= \sum_{s=1}^{S}\sum_{j=1}^{J} 1_{\{\sigma_n = \sigma^*, (\gamma_n^s(I_j^1)\gamma_n^s(I_j^2)) = (\alpha\beta)\}} \\ -S\frac{\sum_a \prod_{j=1}^{J} P(\sigma_n(j), a(I_j^1)a(I_j^2)) \sum_{j=1}^{J} 1_{\{\sigma_n = \sigma^*, (a(I_j^1)a(I_j^2)) = (\alpha\beta)\}}}{\sum_b \prod_{j=1}^{J} P(\sigma_n(j), b(I_j^1)b(I_j^2))} & \end{aligned} \tag{3.52}$$

If we let

$$H(\gamma_n) = \sum_{s=1}^{S}\sum_{j=1}^{J} 1_{\{\sigma_n = \sigma^*, (\gamma_n^s(I_j^1)\gamma_n^s(I_j^2)) = (\alpha\beta)\}} \tag{3.53}$$

then equation (3.52) becomes

$$\frac{\partial}{\partial P(\sigma^*, \alpha\beta)} \log(L(\theta|x)) = H(\gamma_n) - \sum_a H(a) \frac{\prod_{j=1}^{J} P(\sigma_n(j), a(I_j^1)a(I_j^2))}{\sum_b \prod_{j=1}^{J} P(\sigma_n(j), b(I_j^1)b(I_j^2))} \tag{3.54}$$

If we let

$$\Pi(\gamma_n) = \frac{1}{Z}\prod_{j=1}^{J} P(\sigma_n(j), \gamma_n(I_j^1)\gamma_n(I_j^2)) \tag{3.55}$$

Then equation (3.54) can be written as

$$\frac{\partial}{\partial P(\sigma^*, \alpha\beta)} \log(L(\theta|x)) = H(\gamma_n) - \sum_a H(a)\Pi(a) = H(\gamma_n) - \mathbb{E}(H), \tag{3.56}$$

Where $\mathbb{E}(H)$ is the expected value of H under the distribution Π . We set this equal to zero. Therefore we want to find the value of φ such that

$$H(\gamma_n) = \mathbb{E}(H) \tag{3.57}$$

This has no closed form solution since it is of the form

$$K_{\alpha\beta}\sum e^{b_i x_i} = \sum D_{\alpha\beta} e^{a_i x_i} \tag{3.58}$$

and we need to solve for b.

Taking the partial derivatives with respect to all possible values gives us a system of nonlinear equations. It is necessary to use a numerical technique such as steepest decent to solve this system of non-linear equations. Hence, it is equivalent to find the minimum value of

$$\sum_{\alpha\beta}\left(K_{\alpha\beta}\sum e^{b_i x_i} - \sum D_{\alpha\beta} e^{a_i x_i}\right)^2 \tag{3.59}$$

For an initial approximation to the true solution we can use the MLE that we obtained for basepairs. See equations (3.32), (3.37), and (3.45).

3.4 MLE for a J_c-node

Let IM_c be the interaction matrix for the center bases. For J_c-nodes we let γ be the combination of the letter on the left, center and right. The distribution function for a J_c-node is very similar to (2.15) and is given by

$$P((L,C,R)=(\ell,c,r))=\frac{1}{Z}\prod_{j=1}^{J}P(\sigma(j),\gamma(I_j^1)\gamma(I_j^2))\cdot u\prod_{\#\text{ins}}\tau\cdot v\prod_{\#\text{ins}}\omega\cdot\varepsilon\prod_{\#\text{ins}}\epsilon \tag{3.60}$$

Again, as is the case for C-nodes, the MLE for P is a solution to a system on nonlinear equations given by (3.59)

3.5 Parameter estimates from data

Using ad hoc parameters we make alignments of sequences. See Figure (3.1). From this alignment we use equation (3.37) to estimate the parameters for the cWW family. Table (3.2) shows the new parameters that we estimated. Using these new parameters we re-align the sequences. See Figure (3.2). Table (3.3) shows the parameters obtained from our new alignment

cWW	A	C	G	U
A	0.0211	0.0573	0.0211	0.1556
C	0.0573	0.0211	0.1556	0.0211
G	0.0211	0.1556	0.0010	0.0573
U	0.1556	0.0211	0.0573	0.0211

Table 3.1: Ad hoc Isostericity Matrix for cWW A-U,C-G basepairs

cWW	A	C	G	U
A	0.0032	0.0377	0.0032	0.2059
C	0.0377	0.0032	0.2059	0.0032
G	0.0032	0.2059	0.0037	0.0377
U	0.2059	0.0032	0.0377	0.0032

Table 3.2: Data Matrix for cWW A-U,C-G basepairs

cWW	A	C	G	U
A	0.0025	0.0367	0.0025	0.2080
C	0.0367	0.0025	0.2080	0.0025
G	0.0025	0.2080	0.0037	0.0367
U	0.2080	0.0025	0.0367	0.0025

Table 3.3: New Data Matrix for cWW A-U,C-G basepairs

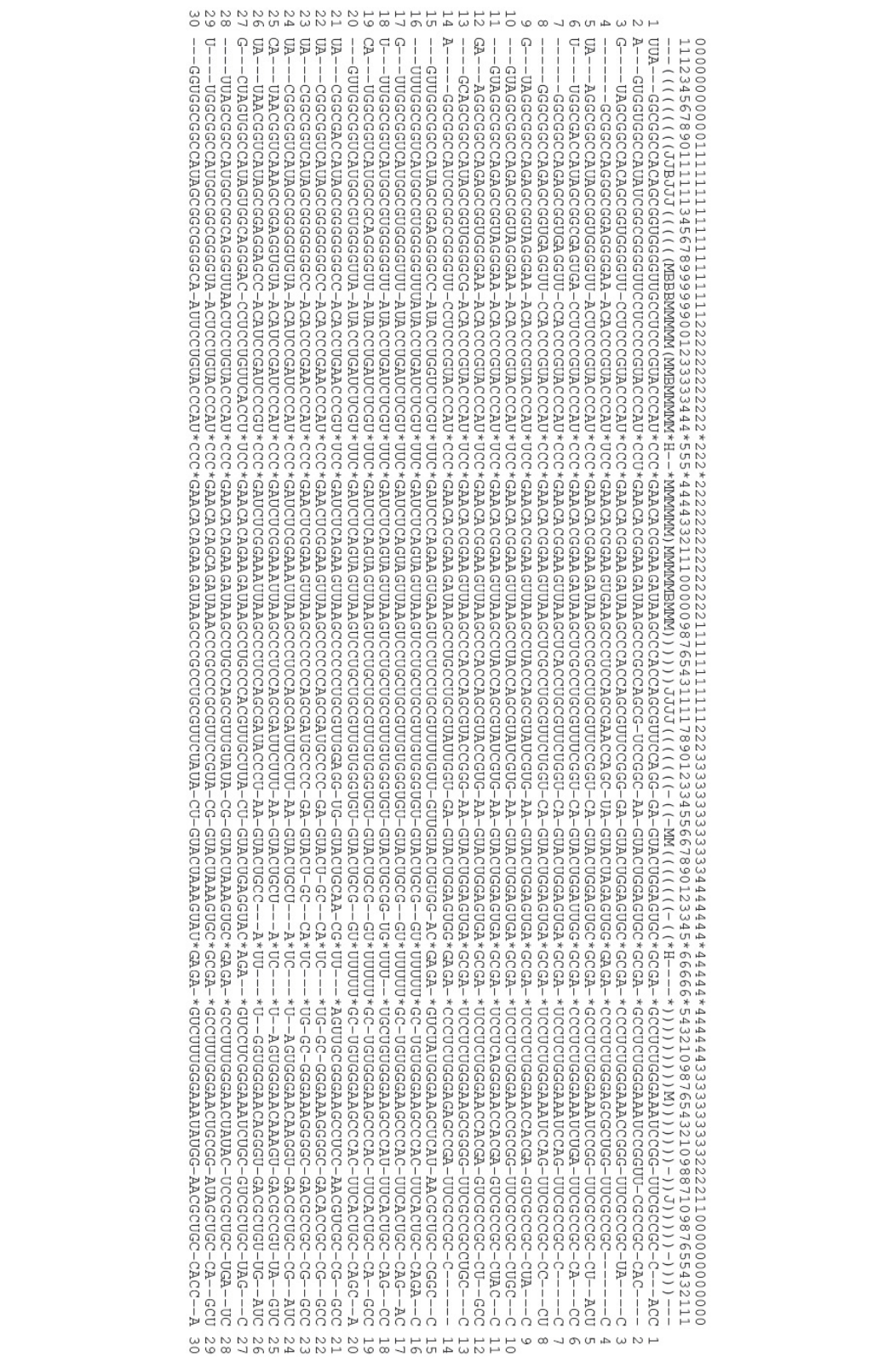

Figure 3.1: Alignment of 5S RNA sequences using ad hoc parameters given by Table (3.1)

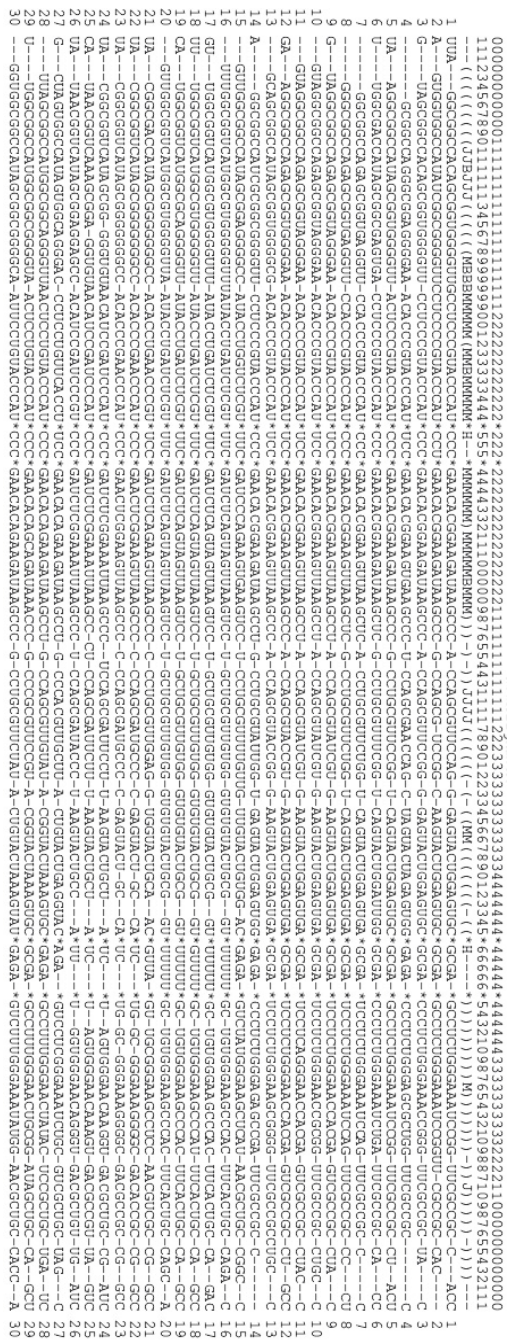

Figure 3.2: Alignment of 5S RNA sequences using estimated parameters given by Table (3.2)

CHAPTER 4

Motif Searching

Recall in Chapter 1, we have a crystal structure with many nucleotides. In it, there are repeated structures such as helices and kink-turns. Helices are common and well understood, but kink-turns are not. Once you find one, how can you find others that are geometrically similar?

Our problem is that we are given a geometric object in 3-space that consists of m points that we want to find other objects of similar shape but with arbitrary location and orientation in 3D space. Let $(x_1, \ldots, x_m)$ be the coordinates in 3-space of all m points. Let $D_{ij} = \|x_i - x_j\|$ be the pairwise distance matrix. Then, D_{ij} gives the Euclidean distance between x_i and x_j in Angstroms. In our first approach to the problem we set cutoffs/tolerances for the distances between two points. We considered two points (α, β) to match (x_i, x_j) if the distance between α and β is within $D_{ij} \pm T$. We defined *pairwise screening matrices* $A_{i,j}$ as

$$A_{i,j}(\alpha, \beta) = \begin{cases} \|\alpha - \beta\| & \text{if } D_{ij} - T \leq \|\alpha - \beta\| \leq D_{ij} + T \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

We want to find all $(\alpha_1, \ldots, \alpha_m)$ such that $A_{i,j}(\alpha_i, \alpha_j) > 0$ for all $1 \leq i < j \leq m$. These are sets of points whose mutual distances fall within tolerance T of the mutual distances of $x_1, \ldots, x_m$.

4.1 Our first approach

To construct candidates, we start by first constructing triangles. Let

$$I_{1,2} = \{(\alpha_1, \alpha_2) | A_{1,2}(\alpha_1, \alpha_2) > 0\}. \tag{4.2}$$

Then $I_{1,2}$ is a $k \times 2$ matrix that contains all the candidates for the sub-motif consisting of the first two bases. Next, for each row of $I_{1,2}$ we find the set of all α_3 such that $A_{1,3}(I_{1,2}(v,1), \alpha_3) > 0$ and $A_{2,3}(I_{1,2}(v,2), \alpha_3) > 0$. This gives us

$$I_{1,2,3} = \{(\alpha_1, \alpha_2, \alpha_3) | A_{1,2}(\alpha_1, \alpha_2) > 0, A_{1,3}(\alpha_1, \alpha_3) > 0, A_{2,3}(\alpha_2, \alpha_3) > 0\} \tag{4.3}$$

which is a matrix that contains all the candidate sub-motifs consisting of the first three bases. We then do the same procedure to obtain sub-motif candidates of the first two bases and the n^{th} base. This gives us the sets

$$I_{1,2,n} = \{(\alpha_1, \alpha_2, \alpha_n) | A_{1,2}(\alpha_1, \alpha_2) > 0, A_{1,n}(\alpha_1, \alpha_n) > 0, A_{2,3}(\alpha_2, \alpha_n) > 0\} \tag{4.4}$$

Next, we want to make four base sub-candidates. To do this, we take $I_{1,2,3}$ and $I_{1,2,4}$ and for each row in $I_{1,2,3}$ we take the first two entries in the row and see if we can find the same pair in the first two entries of some row of $I_{1,2,4}$. Assume that the first two entries in row v_1 of $I_{1,2,3}$ are the same as the first two entries in row v_2 of $I_{1,2,4}$. We would then check to see if the distance between $I_{1,2,3}(v_1, 3)$ and $I_{1,2,4}(v_2, 3)$ are within the appropriate tolerances. That is, we check to see that $A_{3,4}(I_{1,2,3}(v_1, 3), I_{1,2,4}(v_2, 3)) > 0$. If the distance is within the tolerance than we have a four base sub-candidate, namely $(I_{1,2,3}(v_1, :), I_{1,2,4}(v_2, 3))$. We then construct a list of these sub-candidates.

$$I_{1,2,3,4} = \{(\alpha_1, \alpha_2, \alpha_3, \alpha_4) | A_{i,j}(\alpha_i, \alpha_j) > 0,\ 1 \leq i < j \leq 4\} \tag{4.5}$$

For a concrete example let

$$I_{1,2,3} = \begin{bmatrix} 2 & 4 & 8 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \\ 3 & 7 & 12 \end{bmatrix} \tag{4.6}$$

$$I_{1,2,4} = \begin{bmatrix} 2 & 3 & 15 \\ 2 & 4 & 17 \\ 3 & 5 & 17 \\ 4 & 9 & 19 \end{bmatrix} \tag{4.7}$$

Then looking at the first row of $I_{1,2,3}$ we get (2,4,8). Next we look to see if (2,4) appears as the first two numbers in any row of $I_{1,2,4}$. We have a match with row two. The second row of $I_{1,2,4}$ is (2,4,17). Now, if $A_{3,4}(8, 17) > 0$ then one of our four-base sub-candidates is given by (2,4,8,17).

Once we have fully constructed $I_{1,2,3,4}$ then we combine it with $I_{1,2,5}$ in the same fashion as before. We again check for matches with the first two entries of each row and if there is a match we then need to check to see if the new candidate falls within the tolerances. In this case we need to check two tolerances, the one between positions 3,5 and 4,5. If both distances are within tolerances than we add it to $I_{1,2,3,4,5}$. We continue this procedure until all triangles have been combined and we have constructed

$$I_{1,2,\ldots,m} = \{(\alpha_1, \alpha_2, \ldots, \alpha_m) | A_{i,j}(\alpha_i, \alpha_j) > 0,\ 1 \leq i < j \leq m\} \tag{4.8}$$

4.2 Current way of finding pentahedra

Here we describe the current way of finding pentahedra. Matlab code can be found in section 5.4. As before we have a geometric object in 3-space that consists of five points, and we want to find other objects of similar shape. In other words, we want to find all $(\alpha_1, \ldots, \alpha_m)$

such that $A_{i,j}(\alpha_i, \alpha_j) > 0$ for all $1 \leq i < j \leq 5$, where $A_{i,j}$ is given by equation(4.1).

We now find all the nonzero entries in $A_{1,3}$ and $A_{2,3}$ that share a common pair from $I_{1,2}$. See equation (4.2). That is, for each n, $1 \leq n \leq N_1 = |I_{12}(\cdot, 1)|$, we want to find all the values α such that $A_{1,3}(I_{1,2}(n,1), \alpha)$ and $A_{2,3}(I_{1,2}(n,2), \alpha)$ are nonzero. We do the same thing for $A_{1,4}, A_{2,4}$ and $A_{1,5}, A_{2,5}$. Let

$$k_n^m = \{\alpha | A_{1,m}(I_{1,2}(n,1), \alpha),\ A_{2,m}(I_{1,2}(n,2), \alpha) > 0\},\ n \in \{1, \ldots, N_1\},\ m \in \{3,4,5\} \quad (4.9)$$

be the set of all α such that the distance between α and the points in $I_{1,2}(n,:)$ are within tolerances and therefore forms triangles.

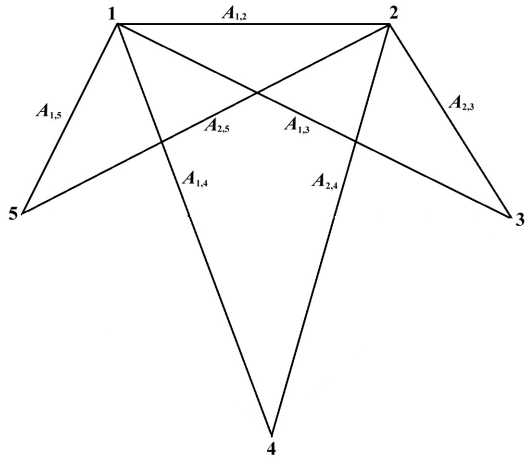

Figure 4.1: Formation of 3 base triples

We have now essentially created three triangles. See figure (4.1). We now want to combine the triangles formed by $A_{1,2}, A_{1,3}, A_{2,3}$ and $A_{1,2}, A_{1,4}, A_{2,4}$. To do this we look at all the nonzero values of $A_{3,4}$ restricted to the sets k_n^3 and k_n^4. This will give us a set of tetrahedra formed from the bases 1,2,3 and 4. See figure (4.2). Therefore, we set

$$I_{3,4} = \{(\alpha, \beta) | A_{3,4}(\alpha, \beta) > 0, \alpha \in k_n^3, \beta \in k_n^4\} \quad (4.10)$$

and $N_2 = |I_{3,4}(\cdot, 1)|$.

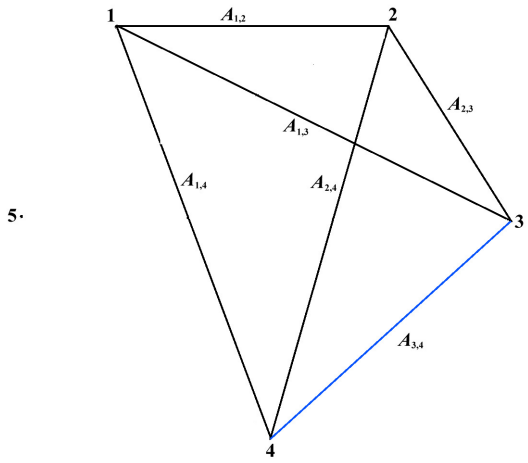

Figure 4.2: Creating tetrahedra

Finally, we want to finish creating our pentahedra. Therefore, we need to find all the α such that $\alpha \in k_n^5$. This restricts the number of possible candidates for the fifth position to ones that satisfy the tolerance requirements to be paired with one and two. Now we need to restrict α to the ones that fit with positions three and four. To do this we need to look at $A_{3,5}$ and $A_{4,5}$. Equation (4.10) gives us all the candidates for positions three and four that we need to consider so we only need to look at α that satisfy $A_{3,5}(I_{3,4}(p, 1), \alpha) > 0$ and $A_{4,5}(I_{3,4}(p, 2), \alpha) > 0$ for $p \in \{1, 2, 3, \ldots, N_2\}$. Therefore we let

$$r_{n,p} = \{\alpha | A_{3,5}(I_{3,4}(p, 1), k_n^5(\alpha)) > 0 \text{ and } A_{4,5}(I_{3,4}(p, 2), k_n^5(\alpha)) > 0\} \tag{4.11}$$

Then for $q \in \{1, 2, 3, \ldots, |r_{n,p}|\}$, if the set $r_{n,p}$ is nonempty then the candidates are given by

$$[I_{1,2}(n, 1)\ I_{1,2}(n, 2)\ I_{3,4}(p, 1)\ I_{3,4}(p, 2)\ r_{n,p}(q)] \tag{4.12}$$

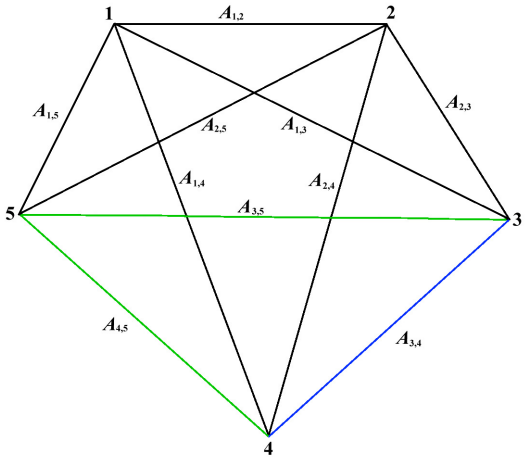

Figure 4.3: Creating pentahedra

4.3 Maximum number of candidates

We would like to have an upper bound for the number of candidate motifs so we can give an upper bound on the execution time of the search program. If the query motif has m bases then clearly n^m is an upper bound for the number of candidates, where n is the number of bases in the RNA molecule. This bound is clearly very far from the true number of plausible candidates because it includes many candidates whose nucleotides are physically much too far apart to be interacting. Assume there is a minimum distance between nucleotides. Then imagine constructing a sphere that is large enough to encase the query motif. Since we are assuming that there is a minimum distance between bases there are only so many bases that could fit in the sphere. Let b be the maximum number of bases that can fit in the sphere. Now, let's see how many candidate motifs we can find in this sphere. The number of ways to pick m objects from b is $b(b-1)\cdots(b-m+1)$. Now, this sphere could be centered on any of the n bases in the molecule. Therefore our bound for the number of candidates is given by $n \cdot b(b-1)\cdots(b-m+1)$ or roughly $n \cdot b^m$. This tells us that the number of candidates only grows linearly in the length n of the molecule.

Table (4.1) gives the value of b for spheres of different radii from the 23S ribosomal RNA

molecule.

Angstroms	Max	Average
5Å	4	2.0
10Å	16	8.8
15Å	35	20.3
20Å	80	39.8
25Å	146	73.2
30Å	214	121.4
35Å	330	182.9

Table 4.1: Values of b for spheres of different radii from the 23S ribosomal RNA molecule.

4.4 The definition of discrepancy

In order to know if our candidates are a "good match" to the query motif we need a measure of how "close" the two motifs are. To do this we define a discrepancy that takes into account the positions of the bases and their relative orientations.

Let m be the number of bases in the query motif. We then define b_i and c_i to be the geometric center of the heavy base atoms of the query and candidate respectively. We define the *fitting error* L as

$$L^2 = \min_R \min_t \sum_{i=1}^{m} w_i \|b_i - R(c_i - t)\|^2 \tag{4.13}$$

where t is the *translation vector*, R is a 3×3 rotation matrix, and w_i are weights, which are usually 1, such that $w_i > 0$ and $\sum_{i=1}^{m} w_i = m$. L gives us a measure of how close the bases are in 3-space.

Now we need to take into account the relative orientations of the bases. Let M_i and N_i be the rotation matrices that take a base in standard orientation to the orientation of base i in the query and candidate respectively. Once the candidate is rotated onto the query motif by R, the rotation matrix $M_i N_i^{-1} R^{-1}$ tells how to rotate base i of the candidate onto base i of the query motif We can then define α_i as the angle of rotation for the rotation matrix

$M_i N_i^{-1} R^{-1}$. Then the *orientation error* A is

$$A = \sqrt{\sum_{i=1}^{m} v_i^2 \alpha_i^2} \tag{4.14}$$

where the v_i are weights that can be adjusted to make the discrepancy more sensitive to the orientation of particular bases. Finally we can define the discrepancy between two motifs as

$$D = \frac{1}{m}\sqrt{L^2 + A^2} \tag{4.15}$$

4.5 Permutation of bases

4.5.1 Why do we need to permute the bases?

In the algorithm used to find the motifs that are similar to a given query motif, the length of the main outside loop is determined by the number of nonzero entries in the matrix $A_{1,2}$. If we could make the outside loop shorter we would expect that the search time would be shorter.

4.5.2 How do we permute the bases?

Let n be the number of nucleotides in the query motif. Let $A_{i,j}$ be the pairwise screening distance matrix between bases i and j. Let $|A_{i,j}|$ be the number of nonzero entries in $A_{i,j}$. We then set $I = [1\ 2\ 3\ \cdots\ n]$ which is the current order of the bases.

Let ℓ_1 and ℓ_2 be such that $|A_{\ell_1,\ell_2}| = \min_{i,j\in\{1,2,\ldots,n\}} |A_{i,j}|$. Next, we switch the positions of 1 and 2 with ℓ_1 and ℓ_2 respectively. Therefore, I becomes

$$I = \begin{bmatrix} \ell_1 & \ell_2 & 3 & 4 & 5 & \cdots & \ell_1 - 1 & 1 & \ell_1 + 1 & \cdots & \ell_2 - 1 & 2 & \ell_2 + 1 & \cdots & n \end{bmatrix} \tag{4.16}$$

Next, we repeat this process except we look for the minimum of $|A_{i,j}|$ where $i, j \neq \ell_1$ and

$i, j \neq \ell_2$. Let ℓ_3 and ℓ_4 be such that $|A_{\ell_3,\ell_4}| = \min_{i,j \in \{1,2,\ldots,n\} \setminus \{\ell_1,\ell_2\}} |A_{i,j}|$. Then I can be written as

$$I = \begin{bmatrix} \ell_1 & \ell_2 & \ell_3 & \ell_4 & 5 & 6 & \cdots & n \end{bmatrix} \tag{4.17}$$

We continue to find the minimum of $|A_{i,j}|$ over smaller and smaller sets until all the bases positions have been permuted. The new order of the bases is given by

$$I = \begin{bmatrix} \ell_1 & \ell_2 & \ell_3 & \ell_4 & \ell_5 & \ell_6 & \cdots & \ell_n \end{bmatrix} \tag{4.18}$$

In the case when the length of the query motif is three or four nucleotides long then this permutation is optimal in the sense that the search time is minimized with respect to our algorithm. To see this note that the search algorithm for the three and four nucleotide case only has two loops in the code, one outer loop whose length is determined by the number of nonzero entries in $A_{1,2}$ and an inner loop. Since the order of the bases has no effect on the candidates found, the code inside the inner loop is called the same number of times no matter what permutation is used. Therefore, the only loop that contributes to the computation time is the outer loop. Hence, we want to minimize the length of the outer loop.

4.6 Displaying Candidates

We have written a program to display the candidates in a nice way. It is available with FR3D [9], at `rna.bgsu.edu/FR3D`. It displays the query motif in one window and the candidate in another. There is a small GUI that allows you to select from several options. See the far left of Figure (4.4). Besides displaying the nucleotides, the plot of the query motif displays at the top of the view window the bases that were used and the file which the bases came from. For the candidate motif, in addition to the same information that is displayed for the query motif, the plot number and the total number of candidates are displayed in the lower left hand corner of the view window along with the discrepancy between the current candidate

and the query motif.

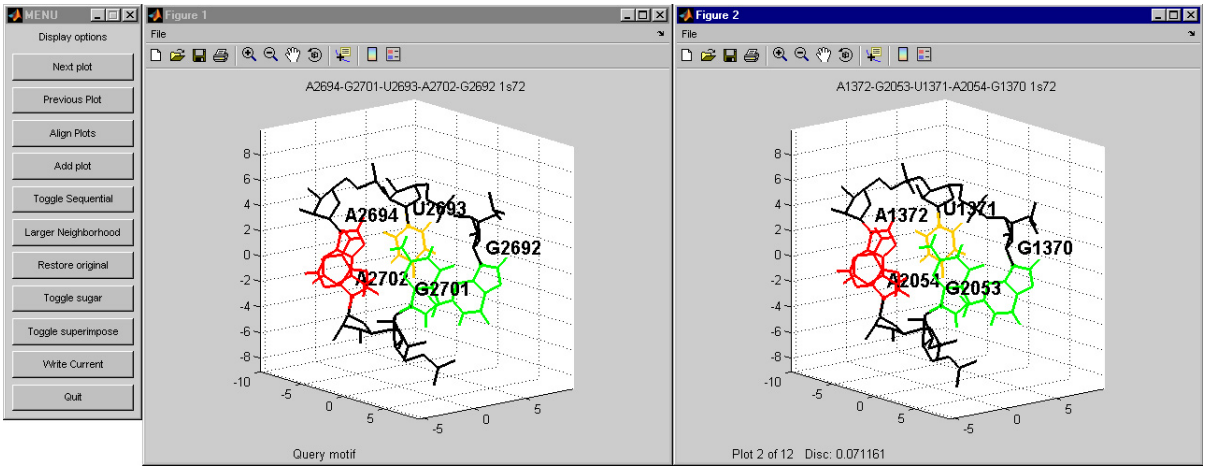

Figure 4.4: Display of candidates using FR3D

By using the Next plot and Previous plot buttons we can cycle through the list of candidates.

The Align Plots button determines the current viewing parameters of the query motif and changes the viewing parameters of the candidate motif to match. This is one way to see the similarities and differences between the two motifs.

The Add plot button opens a new graph window for displaying candidates. By clicking on a graph window we make the current graph active. Then when we press the next or previous plot button only the current graph window will cycle through the candidates. Also, using the align plots button will align all the plots to the query motif no matter how many plots you have displayed.

The Larger Neighborhood button lets us see the candidate motif along with some of the bases that are close to the motif. This is useful if you are searching for a larger motif but you are only using a small number of bases in the search. When the candidates are initially displayed you only see the bases that you searched for. With this option we can essentially zoom out and see the rest of the motif that we didn't include in our search parameters.

The Restore original button returns the candidate to its original display.

The Toggle sugar button allows you to either show the motif with the sugars attached or

not. This gives a nice view of the nucleotides unobstructed by the sugars.

The Toggle Superimpose button uses the current candidate and the query motif and it displays the two motifs superimposed onto one another. See Figure (4.5).

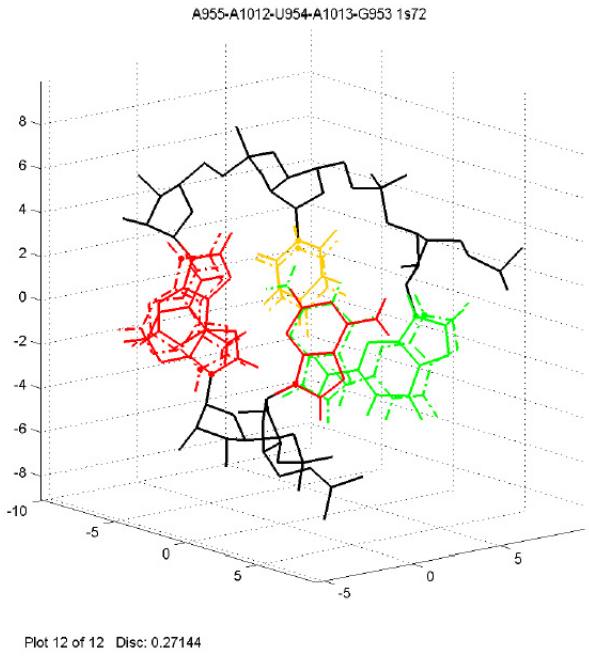

Figure 4.5: Superimposed motifs

The Write current button writes a text file of the bases from the current candidate, its discrepancy, and the PDB file in which the candidate came from. It then displays on the window that the current candidate has been written to a file. If you view other candidates and then come back to the one you had written the display window will display that the current candidate had been written to a file.

CHAPTER 5

Selected Matlab Code

5.1 Matlab code for Cluster Nodes

In the following code i is the start of the subsequence to be parsed and j is the end of the subsequence. Node is our model of the RNA molecule and n is the current node that is being considered.

```
case 'Cluster'
 nextnum = Node(n).nextnode;            % number of next node
        % if the next node is an alternative node then we look to see what is the
        % node that the alternative chose to be the next node.
 if strcmp(Node(nextnum).type,'Alternative')
  nextnum=Node(nextnum).nextnode(P(nextnum,1).Alt(i,j));
 end                                    %
 next    = Node(nextnum);               % next node in the tree
 maxmat = [];                           % to store max values for insertions
 % ---------------------------------- Compute probabilities if we delete
 for c = 1:next.numstates,              % loop through states of next node
  tc = P(nextnum,c).transcode(i,j);     % code for transition prob
```

```
                                        % max probability for children
 maxmat(1,c) = P(nextnum,c).mp(i,j) + next.lP(1,tc);
end                                     % end for c
[mp,cs] = max(maxmat);                  % find the max prob child state
P(n,1).mp(i,j)   = mp;                  % store the maximum probability
P(n,1).next(i,j) = cs;                  % child state of next node
P(n,1).sub(i,j)  = i;                   % start of subsequence for next node
P(n,1).sub(j,i)  = j;                   % end of subsequence for next node
% ---------------------------------- End Computing probabilities if we delete
mp = -Inf;                              % Set max probability to - infinity
NL = length(Node(n).Left(1,:));         % number of bases on the left
NR = length(Node(n).Right(1,:));        % number of bases on the right
aa = i;                                 % Initialize variable
bb = j;                                 % Initialize variable
cs = 1;                                 % Initialize variable
lli = 1;                                % Initialize variable
rri = 1;                                % Initialize variable
for li = 1:length(Node(n).Left(:,1)),       % left interaction possibilities
 a = i + Node(n).Left(li,NL);               % first element for child
 for ri = 1:length(Node(n).Right(:,1)),     % right interaction possibilities
  b = j - Node(n).Right(ri,1);              % last element for child
  if a <= b,                                % room for child too
   lc = Code(i - 1 + Node(n).Left(li,:));   % codes of bases on the left
   rc = Code(j + 1 - Node(n).Right(ri,:));  % codes of bases on the right
   co = [lc rc];                            % codes of all bases to consider
   if max(co) < 5,                          % letters include . for "hairpin"
    S = Node(n).LLIP(li) + Node(n).LRIP(ri);% total score of interactions
```

```
    for k = 1:length(Node(n).IBases(:,1)),  % loop through interactions
     S = S+Node(n).Score(co(Node(n).IBases(k,1)),co(Node(n).IBases(k,2)),k);
    end                                     % end for k
    for c = 1:next.numstates,               % loop through states of next node
     tc = P(nextnum,c).transcode(a,b);      % code for transition prob
     maxmat(c) = P(nextnum,c).mp(a,b) + next.lPIns(tc);
                                            % prob with this specific insertion
                                            % no interaction between this motif
                                            % and the next node
    end                                     % end if c
    [y,mc] = max(maxmat);  % max probability for child
    S = S + y;             % total score for pairwise interactions
    if S > mp,             % Is new max higher
     mp  = S;              % Set max probability
     cs  = mc;             % child state of next node
     aa  = a;              % start of subsequence for next node
     bb  = b;              % end of subsequence for next node
     lli = li;             % which interaction pattern to use for the left
     rri = ri;             % which interaction pattern to use for the right
    end                    % end if S > mp,
   end                     % end if max(co) < 5,
  end                      % end if a <= b,
 end                       % end for ri
end                        % end for li
P(n,2).mp(i,j)   = mp;     % store the maximum probability
P(n,2).next(i,j) = cs;     % child state of next node
P(n,2).sub(i,j)  = aa;     % start of subsequence for next node
```

```
 P(n,2).sub(j,i)  = bb;     % end of subsequence for next node
 P(n,2).mp(j,i)   = lli;    % which interaction pattern to use for the left
 P(n,2).next(j,i) = rri;    % which interaction pattern to use for the right
```

5.2 Matlab code for JunctionCluster Nodes

The inputs for this program are the same as the Cluster Node. See section (5.1).

```
case 'JunctionCluster'
 nextnum = Node(n).nextnode;       % Number of next node
 next1    = Node(nextnum(1));      % Next node in the tree, left side of junction
 next2    = Node(nextnum(2));      % Next node in the tree, right side of junction
 maxmat = [];                      % To store max values for insertions
 mp = -Inf;                        % Set max probability to -infinity
 NL = length(Node(n).Left(1,:));   % Number of bases on the left
 NR = length(Node(n).Right(1,:));  % Number of bases on the right
 NM = length(Node(n).Middle(1,:)); % Number of bases on the middle
 aa = i;                           % Initialize variable
 bb = j;                           % Initialize variable
 cs = 1;                           % Initialize variable
 lli = 1;                          % Initialize variable
 rri = 1;                          % Initialize variable
 cc  = i+1;                        % Initialize variable
 mmi = 1;                          % Initialize variable
 nml1=Node(n).minl(1);             % Fix the minimum value of m
 nml2=Node(n).minl(2);             % Fix the maximum value of m
 mm  = i+nml1;                     % Initialize mm
 for li = 1:length(Node(n).Left(:,1)),     % Left interaction possibilities
```

```
a = i + Node(n).Left(li,NL);              % First element for child 1
for ri = 1:length(Node(n).Right(:,1)),    % Right interaction possibilities
 b = j - Node(n).Right(ri,1);             % Last element for child 2
 for m=(i+nml1):(j-nml2),                 % Bifurcation location; last for child 1
  for mi = 1:length(Node(n).Middle(:,1)),% Middle interaction possibilities
   c = m + Node(n).Middle(mi,NM);         % first element for child 2
   if a <= m-1 & c<=b,                    % Make sure order is correct
                                          % Max probability for children
    maxprob= P(nextnum(1),1).mp(a,m-1) + P(nextnum(2),1).mp(c,b);
    lc = Code(i - 1 + Node(n).Left(li,:));   % Codes of interacting bases
                                             % on the left
    mc = Code(m - 1 + Node(n).Middle(mi,:)); % Codes of bases on the middle
    rc = Code(j + 1 - Node(n).Right(ri,:));  % Codes of bases on the right
    co = [lc mc rc];                         % Codes of all bases to consider
    if max(co) < 5,                          % Letters include . for "hairpin"
                                             % Insertion probabilities
     S = Node(n).LLIP(li) + Node(n).LMIP(mi) + Node(n).LRIP(ri);
     for k = 1:length(Node(n).IBases(:,1)),  % Loop through interactions
                                             % total score or pairwise interactions
      S = S+Node(n).Score(co(Node(n).IBases(k,1)),co(Node(n).IBases(k,2)),k);
     end                           % end for k
     S = S + maxprob;              % Maximum Probability
     if S > mp,                    % Is New max higher
      mp  = S;                     % Set Max probability
      aa  = a;                     % start of subsequence for 1st next node
      mm  = m-1;                   % end of subsequence for 1nd next node
      cc  = c;                     % start of subsequence for 2nd next node
```

```
        bb  = b;                    % end of subsequence for 2nd next node
        lli = li;                   % Which interaction pattern to use for left
        rri = ri;                   % Which interaction pattern to use for right
        mmi = mi;                   % Which interaction pattern to use for middle
       end                          % end if S > mp,
      end                           % end if max(co) < 5,
     end                            % end if a <= m-1 & c<=b,
    end                             % end mi
   end                              % end for m
  end                               % end for ri
 end                                % end for li
 P(n,1).next(i,j) = 0;              % child state of next node
 P(n,1).next(j,i) = 0;              % child state of next node
 P(n,1).sub(i,j)  = aa;             % start of subsequence for 1st next node
 P(n,1).sub(j,i)  = bb;             % end of subsequence for 2nd next node
 P(n,1).mp(i,j)   = mp;             % store the maximum probability
 P(n,1).mp(j,i)   = lli;            % Which interaction pattern to use for left
 P(n,1).rmi(i,j)  = rri;            % Which interaction pattern to use for right
 P(n,1).rmi(j,i)  = mmi;            % Which interaction pattern to use for middle
 P(n,1).sub2(i,j) = cc;             % start of subsequence for 2nd next node
 P(n,1).sub2(j,i) = mm;             % end of subsequence for 1nd next node
```

5.3 Matlab code for parsing Alternatives

```
case 'Alternative'
  nextnode = Node(n).nextnode;      % List of start nodes for each alternative
  numAlt=length(nextnode);          % Number of alternatives
```

```
AltToUse=1;                          % Initialize choice of alternative
mp=-inf;                             % initialize maximum probability
state=1;                             % initialize state
for c=1:numAlt                       % Loop through alternatives
  switch Node(nextnode(c)).type,     % Check to see if the next node
   case {'Basepair','Cluster'}       %  is a basepair or a Cluster
   ss=2;                             %  if it is there are 2 states to consider
   otherwise                         %  if not
   ss=1;                             %  there is only one
  end                                % end for c
  for s=1:ss                         % loop through the states for the next node
    m=P(nextnode(c),s).mp(i,j);      % Use the prior distribution on alternatives
    if m>mp                          % if new max is larger
      mp=m;                          % set new max probability
      AltToUse=c;                    % Alternative to use
      state=s;                       % Sate node is in
    end                              % end if m>mp
  end                                % end for s
end                                  % end for c
P(n,1).mp(i,j)   = mp;               % store the maximum probability
P(n,1).next(i,j) = state;            % store state of next node
P(n,1).sub(i,j)  = i;                % start of subsequence for next node
P(n,1).sub(j,i)  = j;                % end of subsequence for next node
P(n,1).Alt(i,j)  = AltToUse;         % which Alternative to use
```

5.4 Matlab code for finding pentahedra

We want $A(i,j) > 0$, $B(i,k) > 0$, $C(j,k) > 0$, $D(i,m) > 0$, $E(j,m) > 0$, $F(k,m) > 0$, $G(i,r) > 0$, $H(j,r) > 0$, $I(k,r) > 0$, $J(m,r) > 0$.

```
function [TList,count]=Case5(Cutoff,TList,count,A,B,C,D,E,F,G,H,I,J)
  [i,j]=find(A);                          % Find non-zeros of A
  for n=1:length(i),                      % Loop through pairs
    in = i(n);                            % Fix nucleotide corresponding to 1
    jn = j(n);                            % Fix nucleotide corresponding to 2
    k = find(B(in,:) .* C(jn,:));         % Find first triangle
    m = find(D(in,:) .* E(jn,:));         % Find second triangle
    [kk,mm]  = find(F(k,m));              % Form tetrahedra
    r = find(G(in,:) .* H(jn,:));         % Find third triangle
    for p = 1:length(kk),                 % Loop through Tetrahedra
      kkkp = k(kk(p));                    % Fix nucleotide corresponding to 3
      mmmp = m(mm(p));                    % Fix nucleotide corresponding to 4
      SSkm= A(in,jn) + B(in,kkkp) + C(jn,kkkp) + D(in,mmmp) + E(jn,mmmp)...
            + F(kkkp,mmmp);               % Calculate partial matching score
      if SSkm<Cutoff(4)                   % verify we haven't exceeded cutoff
        rr=find(I(kkkp,r) .* J(mmmp,r)); % Find fifth base
        for q = 1:length(rr),             % Loop through Pentahedra
          rrrq=r(rr(q));                  % Fix nucleotide corresponding to 5
          SSrs = SSkm + G(in,rrrq)+H(jn,rrrq)+I(kkkp,rrrq)+J(mmmp,rrrq);
          if SSrs<Cutoff(5)               % verify we haven't exceeded cutoff
            count = count + 1;            % Candidate found, add to count
            TList(count,:) = [in jn kkkp mmmp rrrq]; % add to list of candidates
          end                             % end if SSrs<Cutoff(5)
```

```
        end                                   % end for q
      end                                     % end if SSkm<Cutoff(4)
  end                                         % end for p
end                                           % end for n
```

5.5 Matlab code for permuting the bases

we let

$$PS = \begin{pmatrix} 0 & A'_{2,1} & A'_{3,1} & \cdots & A'_{m,1} \\ A_{2,1} & 0 & A'_{3,2} & \cdots & A'_{m,2} \\ A_{3,1} & A_{3,2} & 0 & & \vdots \\ \vdots & \vdots & & \ddots & A'_{m,m-1} \\ A_{m,1} & A_{m,2} & \cdots & A_{m,m-1} & 0 \end{pmatrix} \tag{5.1}$$

$$LEN = \begin{pmatrix} |A_{2,1}| & 2 & 1 \\ |A_{3,1}| & 3 & 1 \\ |A_{3,2}| & 3 & 2 \\ \vdots & \vdots & \vdots \\ |A_{m,m-1}| & m & m-1 \end{pmatrix} \tag{5.2}$$

Here all the $A_{i,j}$ are $L \times L$ pairwise screening matrices.

```
function [PS,Perm]=sFindPermutation(PS,LEN);
  n=length(PS);              % determine how many bases; PS is mxm block matrix
  Loc=1;                     % which base we are considering now
  i=1:n;                     % Identify permutation
  while Loc+1<n              % Continue if we haven't permuted all the bases
    [a,b]=sortrows(LEN,1);   % Sort by the number of non-zero entries
    [LEN,i]=MakeMinFirst(LEN,b,i,Loc); % Permute two bases
    Loc=Loc+2;               % Two bases have been permuted, increment by two
```

```
  end                          % end while Loc+1<n
  PS=PS(i,i);                  % Permute the bases
  Perm=i;                      % Save the permutation
%-------------------------------------------------
function [LEN,i]=MakeMinFirst(LEN,b,i,Loc)
  k=find(i==LEN(b(1),2));      % Find the position of the ith number
  t=i(Loc);                    %|
  i(Loc)=LEN(b(1),2);          %|-Switch first number
  i(k)=t;                      %|
  k=find(i==LEN(b(1),3));      % Find the position of the ith number
  t=i(Loc+1);                  %|
  i(Loc+1)=LEN(b(1),3);        %|-Switch second number
  i(k)=t;                      %|
                               % remove rows of LEN that a permutation
                               %   has been determined for
  c=find(LEN(:,2)~=LEN(b(1),3) & LEN(:,2)~=LEN(b(1),2)
       & LEN(:,3)~=LEN(b(1),3) & LEN(:,3)~=LEN(b(1),2));
  LEN=LEN(c,:);                % return the bases that still need permuted
```

BIBLIOGRAPHY

[1] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis*. Cambridge, 2003.

[2] Alexander Isaev. *Introduction to mathematical methods in bioinformatics*. Universitext. Springer-Verlag, Berlin, 2004.

[3] D.J. Klein, P.B. Moore, and T.A. Steitz. The roles of ribosomal proteins in the structure, assembly and evolution of the large ribosomal subunit. *Journal of Molecular Biology*, 2004. PDB ID: 1s72.

[4] K. Lari and S.J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speach and Language*, 1990.

[5] N. Leontis, J. Stombaugh, and E. Westhof. The non-watson-crick base pairs and their associated isostericity matrices. *Nucleic Acids Research*, 2002.

[6] A. Mokdad. *Developing tools for RNA structural allignment*. PhD thesis, Bowling Green State University, 2006.

[7] D. Mount. *Bioinformatics*. Cold Spring Harbor Laboratory Press, 2001.

[8] E. Rivas and S. Eddy. The language of rna: A formal grammar that includes pseudoknots. *Bioinformatics*, 2000.

[9] M. Sarver, C. Zirbel, J. Stombaugh, A. Mokdad, and N. Leontis. Fr3d: Finding local and composite recurrent structural motifs in rna 3d structures. *to appear in Journal of Mathematical Biology.*

[10] Gerhard Winkler. *Image analysis, random fields and Markov chain Monte Carlo methods*, volume 27 of *Applications of Mathematics (New York)*. Springer-Verlag, Berlin, second edition, 2003. A mathematical introduction, With 1 CD-ROM (Windows), Stochastic Modelling and Applied Probability.