2011-01-01

Distributional Properties of Inversions and Segmentation Algorithms for RNA Sequences

Sameera Dhananjaya Viswakula
University of Texas at El Paso, sdviswakula@miners.utep.edu

DISTRIBUTIONAL PROPERTIES OF INVERSIONS AND

SEGMENTATION ALGORITHMS FOR RNA SEQUENCES

SAMEERA DHANANJAYA VISWAKULA

Department of Mathematical Sciences

APPROVED:

Ming-Ying Leung, Chair, Ph.D.

Kyle L. Johnson, Ph.D.

Naijun Sha, Ph.D.

Amy Wagler, Ph.D.

Benjamin C. Flores, Ph.D.
Acting Dean of the Graduate School

to my

MOTHER, FATHER and BROTHER

with love

DISTRIBUTIONAL PROPERTIES OF INVERSIONS AND SEGMENTATION ALGORITHMS FOR RNA SEQUENCES

by

SAMEERA DHANANJAYA VISWAKULA

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Mathematical Sciences

THE UNIVERSITY OF TEXAS AT EL PASO

August 2011

Acknowledgements

I would like to thank my graduate adviser and thesis adviser, Dr. Ming-Ying Leung for her enormous guidance and support for my thesis and her parental care through out my stay here in El Paso. I would also like to thank my committee members, Dr. Johnson, Dr. Sha, and Dr. Wagler, for their valuable comments and suggestions.

My gratitude also goes to Dr. Staniswalis for her guidance from the beginning. I would also like to thank Dr. Rosen and Dr. Moschopoulos for their valuable support throughout the MS program. I also extend my appreciation to Maria Barraza, for her workshops on LaTeX. I would like to thank the Department of Mathematical Sciences and the Mathematics and Statistics faculty for all their support.

I also want to thank Leonel Saldivar, Bioinformatics Programmer Analyst, for his continuous support on my thesis, Peter Kelly for his modified version of the Palindrome program, and Rahulsimham Vegesna for his help on consistency testing of my algorithms. Special thanks goes to my parents for their understanding and for their kind support, and especially for connecting with me almost every day through Skype™.

I would like to mention all my Statistics, Mathematics, and Bioinformatics colleagues who were with me all the time and especially my office mates for their co-operation.

NOTE: This thesis was submitted to my Supervising Committee on the July 6, 2011.

Abstract

Ribonucleic acid (RNA) is a long single stranded molecule made up of four types of nucleotide bases: Adenine (A), Cytosine(C), Guanine (G) and Uracil (U). It folds back on itself and forms C-G and A-U complementary base pairs. The set of such hydrogen-bonded pairs in an RNA molecule is called its secondary structure. Knowing the secondary structure of RNA is useful for understanding its biological function. Prediction of RNA secondary structure from the nucleotide sequence has been an important bioinformatics problem for over two decades.

The work in this thesis is motivated by the need to improve the secondary structure prediction accuracy and efficiency for long RNA molecules. It involves investigating the distribution of inversions in random nucleotide sequences. An inversion is a string of nucleotide bases in an RNA sequence followed closely by its inverted complementary sequence downstream. It is the essential element to build any secondary structure. In this study, I focused on a random variable representing the number of inversions in an independent and identically distributed (i.i.d.) letter sequence, sampled from the nucleotide alphabet {A, C, G, U} with base composition $\{p_A, p_C, p_G, p_U\}$. I derived a recursive expression for calculating its mean, obtained simulated values for its variance, and demonstrated that this random variable can be reasonably approximated by a Poisson random variable in a range of inversion parameter values.

Predicting RNA secondary structure is a complicated process. It requires so much computer time and memory that often makes it impractical to perform any detailed predictions for sequences only several hundred bases long. Yet, there exist RNA molecules (e.g., in some viral genomes) of biological interest that contain over a thousand bases. In order to overcome the limitations in computing resources, Taufer et al. (2008) developed an ap-

proach by the grid computing technology. The idea of this approach is to segment a long RNA sequences into smaller chunks and send them to different computers on the grid for individual predictions. Then, the individual predictions are assembled to give the prediction for the original molecule. I developed two algorithms for segmenting a long RNA sequence into small chunks. Both algorithms attempt to identify areas in the sequence with high concentration of inversions and preserve these areas within a single chunk in order to reduce the information loss. The effect of these two segmentation algorithms on secondary structure prediction accuracy is tested on a set of data from the Rfam database of RNA sequences with known secondary structures.

Contents

Chapter 1

Introduction

1.1 Background

Researchers and scientists spend billions on laboratory experiments to find cures for viral diseases such as AIDS, SARS, and Influenza. A common fact is that RNA (ribonucleic acid) serves as the genome of many pathogenic viruses. The behavior of viruses are influenced by the 3D structures of various RNA molecules. So researchers and scientists try to identify the 3D structure of these RNA sequences. It is a time consuming and very expensive process, since RNA structures are not stable. However, useful information about an RNA molecule can be gained from knowing its secondary structure, which is the collection of hydrogen bonded base pairs in the molecule. Since the early 1980's, there have been a lot of interest in mathematical and computational approaches to predict the secondary structure of an RNA molecule from its nucleotide sequence information, which is relatively easy to obtain (Markham & Zuker, 2008).

RNA is a random sequence consisting of four nucleotide bases:

- A-Adenine
- C-Cytosine
- G-Guanine
- U-Uracil

Nucleotide base A makes hydrogen bonds with U. Similarly C makes hydrogen bonds with G. As a result, $A-U$, $U-A$, $C-G$ and $G-C$ pairs are formed. These pairs

are called *complementary pairs*. When a stretch of nucleotides is followed closely by its inverted complementary sequence, we say that there is an *inversion*. For example, the complementary sequence of $ACCGUC$ is $UGGCAG$ and its inverted complementary sequence is $GACGGU$. Here ACCGUC and GACGGU are called the *stems* of the inversions and the number of nucleotide bases in each stem is called *stem length*. Most of the time, one or more unpaired bases exist in between these stems, forming a *gap* between two stems. The number of unpaired bases in between the two stems of an inversion is called *gap size*. Figure 1.1 shows an example of inversion with stem length 6 and the gap size 3.

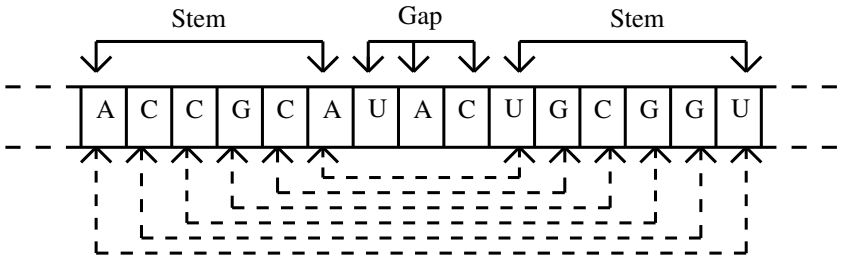


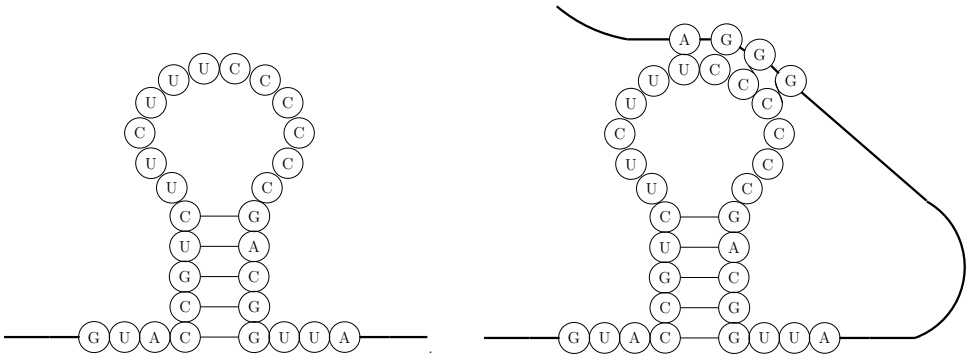
Figure 1.1: Stem length and the gap size of an inversion

There are two basic elements in all naturally occurring RNA secondary structures. They are *stem loops* and *pseudoknots*. An example of each is shown in figure 1.2.

Note that inversions must be presented in both stem-loop and pseudoknots.

1.2 Significance of the Study

Prediction of RNA secondary structure typically requires the minimization of a thermodynamic energy function over the space of all possible distinct sets of complementary pairings that can be formed by the nucleotide sequence. It requires a large amount of memory and computing time, especially when predicting pseudoknots (Rivas & Eddy, 1999) in long RNA sequences with over 100 bases. Due to this reason, only a limited number of pseudoknot types are being predicted and as a result, the overall prediction accuracy is significantly



(a) Stem-loop (b) Pseudoknot

Figure 1.2: Two basic elements in RNA secondary structures

affected. A new method should be employed to utilize the current computing technologies.

Taufer et al. (2008) proposed to use the recently developed grid computing technologies to enhance the capacity of any existing secondary structure prediction algorithm by segmenting a long RNA molecule into smaller chunks, which are distributed to different processors on the grid to be predicted individually. The predicted structures of the smaller chunks are then assembled to form a structure for original larger molecule.

It is anticipated that the accuracy of the final overall structure would depend on how the original long sequence is segmented. Intuitively, one should avoid cutting the sequence within the span of an inversion, including the two stems and the gap between them, that may be part of a secondary structure. The work of this thesis is intended to achieve two goals:

1. To help establish some basic distributional properties for inversions in random RNA sequences in order to guide the design of future sequence segmentation strategy.

2. To test the capabilities of several different segmentation algorithms in preserving the accuracy of the secondary structure prediction method being used.

Chapter 2

Probabilistic Problems Associated with RNA Sequence Segmentation

2.1 Statement of Problems

When designing a segmentation strategy for an RNA secondary structure prediction algorithm, it would be helpful to have an idea about the counts and spans of the inversions and how they are distributed in a random nucleotide sequence with similar base composition. This will allow a rough estimation of how many inversions we are dealing with and how long they are. I will focus on the inversion count and its distribution in this chapter.

Consider the inversions with minimum stem length l and the maximum gap size $\tilde{g}$. For most RNA structures known to occur in nature, it is sufficient to take l and $\tilde{g}$ to be small positive integers (e.g. l=3, $\tilde{g}$=5). Assume that an RNA sequence of length n, is a realization of a sequence of independent and identically distributed random variables, $X_1, X_2, \ldots, X_n$ with values $\{A, C, G, U\}$. For our applications, n is typically in the range of hundreds to thousands. Let Y be the number of such inversions in the sequence. I will consider the following problems

1. What is the expected number of inversions ?
2. What is the variance V[Y] ?
3. Can Y be reasonably approximated by a Poisson random variable ?

We are going to define the locations of an inversion by its center position in the sequence. To make this possible, we define the positions on the RNA sequences as in Figure 2.1.

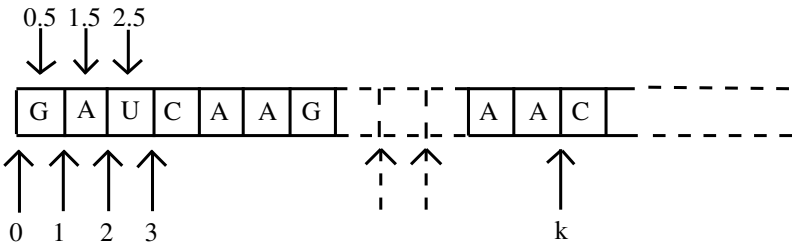


Figure 2.1: Position of the RNA squence

2.2 Expected Number of Inversions

Consider an RNA sequences of length n. Let's assume that nucleotide bases are independent and identically distributed random variables. We define,

$$I_k = \begin{cases} 1 & \text{if the } k^{th} \text{ position is a center of an inversion} \\ 0 & \text{o/w} \end{cases}$$

$$Y = \sum_{k=1}^{n} I_k + \sum_{m=0}^{n-1} I_{m+\frac{1}{2}} \tag{2.1}$$

$$E[Y] = \sum_{k=1}^{n} P(I_k = 1) + \sum_{m=0}^{n-1} P(I_{m+\frac{1}{2}} = 1) \tag{2.2}$$

To calculate the probabilities in (2.2), we first have to consider the probability of complementary pairing. For any two bases in the sequences, they will form a complementary pair if they are $(A, U), (U, A), (C, G)$ and (G, C).

So the probability of having a complementary pair is

$$P_A P_U + P_U P_A + P_C P_G + P_G P_C = 2\left(P_A P_U + P_C P_G\right) \tag{2.3}$$

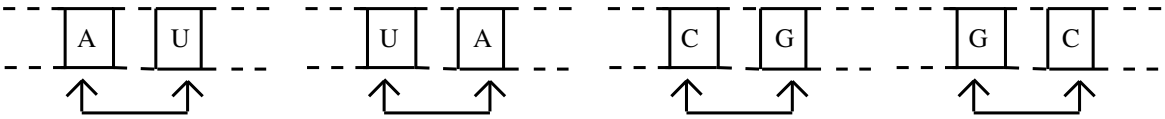


Figure 2.2: Possible bondings between two nucleotide bases

where P_b = probability that nucleotide base b occur in any place in the sequence;
b = A, C, G, U.

$$\begin{aligned} \text{Let } \theta &= 2\left(P_A P_U + P_C P_G\right) \\ l &= \text{minimum stem length of the inversion} \\ g &= \text{exact gap size of the inversion} \end{aligned}$$

Now let us have some examples for calculating $E[Y]$, for $g = 0$ and $g = 1$.

Example 1: $l = 5$ and $g = 0$

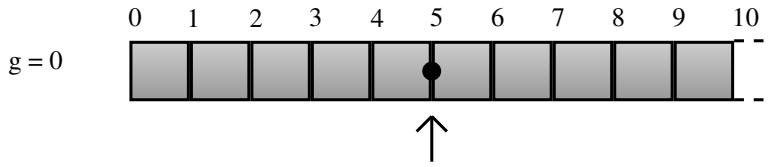


Figure 2.3: l=5, g=0

From Figure 2.3, it can be seen that the probability of having an inversion at position 5 is:

$$\begin{aligned} P(I_5 = 1) &= \theta \times \theta \times \theta \times \theta \times \theta \\ &= \theta^5 \end{aligned}$$

Now consider the expected number of inversions with length 5 and gap size of exactly 0, in anywhere in the sequence of length 20.

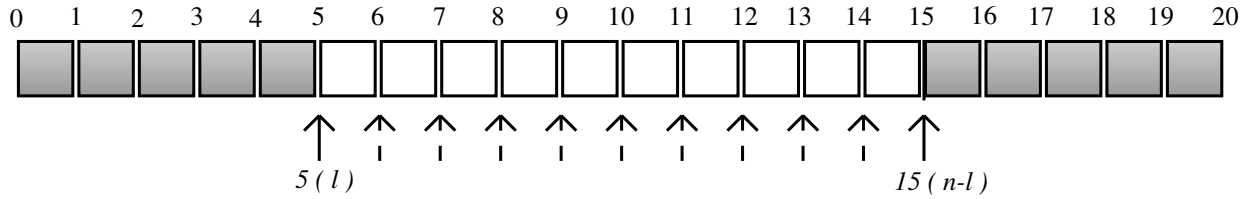


Figure 2.4: l=5, g=0

From Figure 2.4, it can be seen that there can be only 11 ($n-l-l+1 = n-2l+1$) possible positions to place an inversion with $l = 5$ and $g = 0$. Therefore, the expected number of such inversions in a 20 nucleotide sequence is:

$$\begin{aligned}\sum_{k=0}^{20} P(I_k = 1) &= \sum_{k=5}^{15} P(I_k = 1) \text{ (Since all other terms = 0)} \\ &= \theta^5 \times (11) \\ &= 11\theta^5\end{aligned}$$

A general formula can be written for the probability of having inversions of minimum length of l and gap size of 0 in a sequence of length n as follows.

$$\begin{aligned}\sum_{k=0}^{n} P(I_k = 1) &= \sum_{k=l}^{n-l} P(I_k = 1) \\ &= \theta^l \times (n - l - l + 1) \\ &= (n - 2l + 1)\,\theta^l\end{aligned}$$

Example 2: $l = 5$ and $g = 1$.

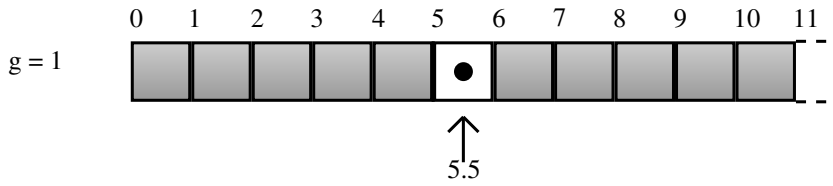


Figure 2.5: l=5, g=1

From Figure 2.5, it can be seen that the probability of having an inversion with $l = 5$ and $g = 1$ at position 5.5 is:

$$\begin{aligned} P(I_{5.5} = 1) &= \theta \times \theta \times \theta \times \theta \times \theta \\ &= \theta^5 \end{aligned}$$

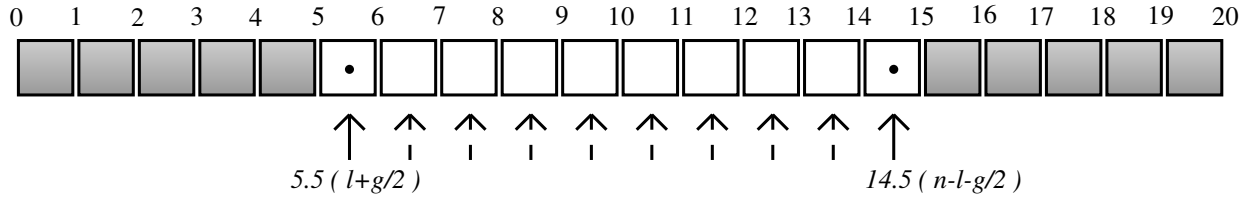


Figure 2.6: l=5, g=1

From Figure 2.6, it can be seen that in a sequence of length 20, there can be 10 ($n - l - g/2 - l - g/2 + 1 = n - 2l - g + 1$) possible center locations of such inversions. Therefore, the expected number is:

$$\begin{aligned}\sum_{m=0}^{20} P(I_{m+\frac{1}{2}}=1) &= \sum_{m=l}^{20-l-1} P(I_{m+\frac{1}{2}}=1) \\ &= \theta^5 \times (10) \\ &= 10\theta^5\end{aligned}$$

A general formula can be written for the expected number of inversions with minimum length of l and gap size of 1 in a sequence of length n as follows.

$$\begin{aligned}\sum_{m=0}^{n-1} P(I_{m+\frac{1}{2}}=1) &= \sum_{m=l}^{n-l-1} P(I_{m+\frac{1}{2}}=1) \\ &= \theta^l \times (n-l-l-1+1) \\ &= (n-2l)\,\theta^l\end{aligned}$$

Example 3: $l = 5$ and $g = 2$.

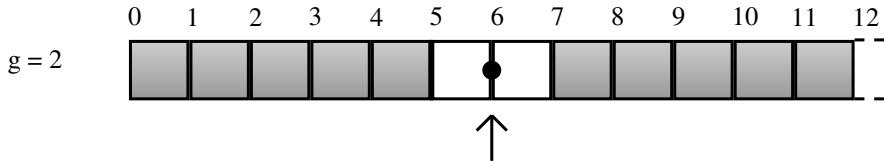


Figure 2.7: l=5, g=2

From Figure 2.7 it can be seen that, the probability of having a center of an inversion in position 6 is;

$$\begin{aligned}P(I_6=1) &= \theta \times \theta \times \theta \times \theta \times \theta \times (1-\theta) \\ &= \theta^5\,(1-\theta)\end{aligned}$$

The factor $(1-\theta)$ is necessary to ensure that the two bases around position b do not pair with each other, so that there is indeed a gap of 2.

Now consider the expected number of inversions with $l = 5$ and $g = 2$, in anywhere in the sequence of length 20. As in the previous examples, it can be shown that

$$\begin{aligned}\sum_{k=0}^{20} P(I_k = 1) &= \sum_{k=l+1}^{20-l-1} P(I_k = 1) \\ &= \theta^5 \times (1-\theta) \times (9) \\ &= 9\theta^5 (1-\theta)\end{aligned}$$

A general formula can be written for the probability of having inversions of minimum length of l and gap size of exactly 2 in a sequence of length n as follows.

$$\begin{aligned}\sum_{k=0}^{n} P(I_k = 1) &= \sum_{k=l+1}^{n-l-1} P(I_k = 1) \\ &= \theta^l \times (1-\theta) \times (n-l-l-2+1) \\ &= (n-2l-1)\,\theta^l (1-\theta)\end{aligned}$$

Similarly, the expected number of inversions for the respective gap sizes anywhere of the sequence are found as follows.

$$
\begin{aligned}
\sum_{m=0}^{n-1} P(I_{m+\frac{1}{2}}=1) &= \sum_{m=l+1}^{n-l-2} P(I_{m+\frac{1}{2}}=1) \quad ; g=3 \\
&= (n-2l-2)\,\theta^l\,(1-\theta) \quad ; g=3 \\
\sum_{k=0}^{n} P(I_k=1) &= \sum_{k=l+2}^{n-l-2} P(I_k=1) \quad ; g=4 \\
&= (n-2l-3)\,\theta^l\,(1-\theta) \quad ; g=4 \\
\sum_{m=0}^{n-1} P(I_{m+\frac{1}{2}}=1) &= \sum_{m=l+2}^{n-l-3} P(I_{m+\frac{1}{2}}=1) \quad ; g=5 \\
&= (n-2l-4)\,\theta^l\,(1-\theta) \quad ; g=5 \\
\sum_{k=0}^{n} P(I_k=1) &= \sum_{k=l+3}^{n-l-3} P(I_k=1) \quad ; g=6 \\
&= (n-2l-5)\,\theta^l\,(1-\theta) \quad ; g=6
\end{aligned}
$$

In general, for a given minimum stem length l and exact gap size $g \geq 2$, we have

$$
\sum_{k=0}^{n} P(I_k=1) + \sum_{m=0}^{n-1} P(I_{m+\frac{1}{2}}=1) = (n-2l-g+1)\theta^l(1-\theta) \tag{2.4}
$$

Let's take $\tilde{g}$ as the maximum gap size of the inversion. Therefore $\tilde{g} \geq g$.

Let $E\left[Y_{l,\tilde{g}}\right]$ = Expected number of inversions of minimum stem length l and maximum gap size g in a given sequence of length n.

From the Equation 2.4, a recursive formula can be written as follows.

$$
\begin{aligned}
E\left[Y_{l,\tilde{g}}\right] &= E\left[Y_{l,\tilde{g}-1}\right] + (n-2l-\tilde{g}+1)\,\theta^l\,(1-\theta)\,; \tilde{g} \geq 2 \\
E\left[Y_{l,1}\right] &= (n-2l+1)\,\theta^l + (n-2l)\,\theta^l \\
E\left[Y_{l,0}\right] &= (n-2l+1)\,\theta^l
\end{aligned} \tag{2.5}
$$

Expected values from formula 2.5 were calculated for a random nucleotide sequence of length 1000 for various l and $\tilde{g}$. Some of the results (for $l = 4$) are included in Table 2.1.

2.3 Variance of Number of Inversions

Variance of the number of inversions, V$[Y]$ can be written as follows.

As earlier, we defined

$$I_k = \begin{cases} 1 & \text{if the } k^{th} \text{ position is a center of an inversion} \\ 0 & \text{o/w} \end{cases}$$

Suppose the number of inversions in the sequence is denoted by Y, then

$$\begin{aligned} V[Y] &= E[Y^2] - E[Y]^2 && (2.6) \\ \text{where } Y^2 &= (\sum_{k=1}^{n} I_k + \sum_{m=0}^{n-1} I_{m+\frac{1}{2}})^2 && (2.7) \\ &= \sum_{k=1}^{n} I_k^2 + \sum_{m=0}^{n-1} I_{m+\frac{1}{2}}^2 && (2.8) \\ &\quad +2(\sum_{k=1}^{n}\sum_{m=0}^{n-1} I_k I_{m+\frac{1}{2}} + \sum_{k_1=1}^{n}\sum_{k_2=1}^{n} I_{k_1} I_{k_2} + \sum_{m_1=0}^{n-1}\sum_{m_2=0}^{n-1} I_{m_1+\frac{1}{2}} I_{m_2+\frac{1}{2}}) \end{aligned}$$

Therefore,

$$\begin{aligned} E[Y^2] &= \sum_{k=1}^{n} P(I_k^2 = 1) + \sum_{m=0}^{n-1} P(I_{m+\frac{1}{2}}^2 = 1) && (2.9) \\ &\quad +2\sum_{k=1}^{n}\sum_{m=0}^{n-1} P(I_k = 1, I_{m+\frac{1}{2}} = 1) + 2\sum_{k_1=1}^{n}\sum_{k_2=1}^{n} P(I_{k_1} = 1, I_{k_2} = 1) \\ &\quad +2\sum_{m_1=0}^{n-1}\sum_{m_2=0}^{n-1} P(I_{m_1+\frac{1}{2}} = 1, I_{m_2+\frac{1}{2}} = 1) \end{aligned}$$

Equations (2.6) and (2.9) imply that in order to calculate V$[Y]$, one will need the possibilities for observing overlapping inversions. There are many possible scenarios of having overlapping inversions. Figure 2.8 shows a possible scenario.

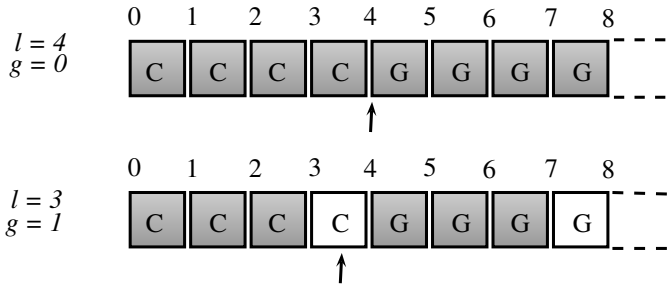


Figure 2.8: Scenario I: Overlapping inversions. The center of the each inversion is denoted by an upward arrow.

In this case, both positions 3.5 and 4 are centers of inversions. It can be seen that one inversion has $g = 0$ while the other inversion has a gap $g = 1$. Another scenario is shown in figure 2.9. In this case, positions 3 and 7 both are the centers of inversions, both with $g = 0$.

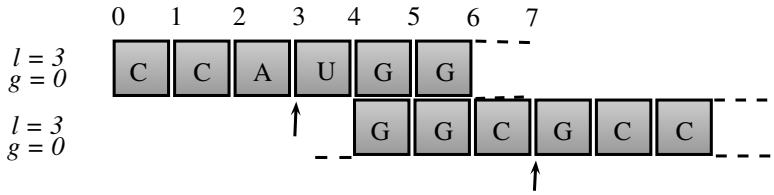


Figure 2.9: Scenario II: Overlapping inversions.The center of the each inversion is denoted by an upward arrow.

There are so many possibilities to be considered, depending on the stem lengths and the gap sizes that it becomes impractical to work out the variance analytically. Therefore, a simulation based method is used to observe the behavior of the variance of number of inversions.

A random nucleotide sequence was generated and the number of inversions of the sequence was counted using a Java program. This Java program is an adapted version of the “Palindrome” program in the *EMBOSS* package (Faller, 1999). *EMBOSS* is "The European Molecular Biology Open Software Suite". It is a free Open Source software analysis package specially developed for the needs of the molecular biology user community. Palindrome

program was modified by Peter Kelly, and we further modified the program to extract the locations of inversions. Number of inversions were identified in random nucleotide sequences and this process was repeated for $100, 200, 400 \ldots 2000$ times and the mean and the variance were obtained for each simulation. These simulations were done for different minimum stem lengths ($l = 3, 4, 5, 6, 7, 8$) and maximum gap sizes ($\tilde{g} = 0, 1, 2, \ldots, 8$). Results for $l = 4$ with maximum gap sizes $\tilde{g} = 0, 1, 2, \ldots, 8$ were shown in Table 2.1 for 100 iterations.

l	$\tilde{g}$	**Theoretical Expected Value**	**Simulated Expected Value ($\bar{X}$)**	**Simulated Variance (S)**	**S/ $\bar{X}$**
4	0	3.879	3.54	3.604	1.018
	1	7.754	7.35	7.523	1.024
	2	10.657	9.66	10.146	1.050
	3	13.558	12.07	11.985	0.993
	4	16.455	14.84	14.277	0.962
	5	19.350	17.46	18.716	1.072
	6	22.241	19.95	19.321	0.968
	7	25.130	22.41	21.780	0.972
	8	28.016	25.29	23.319	0.922

Table 2.1: Theoretical Expected Value, Simulated Expected Value and Simulated Variance for a random nucleotide sequence of length 1000 for $l = 4$ and $\tilde{g} = 0, 1, \ldots, 8$.

2.4 Distributions of the Number of Inversions

Palindromes are symmetrical words of nucleotide sequences in the sense that they read exactly the same as their reverse complementary sequences.They are a special case of inversions with g=0. Leung, *et. al* (2005) proved that in random nucleotide sequences, the occurrences of palindromes, under suitable conditions, can be approximated by a Poisson process. Further, the ratio between the simulated variance and the simulated mean is quite close to 1 (Table 2.1). These led us to formulate the hypothesis that the distribution of the number of inversions can be approximated by a Poisson distribution. To test this hypothesis, we use Kolmogorov-Smirnov Test, The Cramèr-von Mises Test and The Chisquare Goodness-of-fit Test to assess the goodness of fit of the Poisson distribution to the simulated data.

2.4.1 Kolmogorov-Smirnov Test

Kolmogorov and Smirnov developed a statistical procedure to test the goodness of fit of a distribution to a given random sample coming from an unknown continuous distribution. This test uses the maximum vertical distance between the suggested distribution and the empirical distribution of the data (Conover, 1972).

Test Procedure

Data - The data consist of a random sample $X_1, X_2, \ldots, X_n$ of size n associated with some unknown distribution function, denoted by $F(x)$.
Assumptions - The sample is a random sample.
Hypothesis - Let $F^*(x)$ be a completely specified hypothesized distribution function.

$$H_0 \;:\; F(x) = F^*(x) \text{ for all } x \text{ from } -\infty \text{ to } +\infty$$
$$H_1 \;:\; F(x) \neq F^*(x) \text{ for at least one value of } x$$

Test Statistic - Let $S(x)$ be the empirical distribution function based on the random sample $X_1, X_2, ..., X_n$. The test statistics for the above hypothesis is defined as follows.

$$T = \sup_x |F^*(x) - S(x)| \tag{2.10}$$

Decision Rule - Reject H_0 at the level of significance α, if T exceeds the $1 - \alpha$ quantile $w_{1-\alpha}$ as given by Table A14 in Practical Nonparametric Statistics by Conover (1999).

The Kolmogorov goodness of fit test is known to be conservative when the hypothesis distribution function is not continuous. A method for finding the exact critical level for the discrete distributions was derived by Conover (1972). Let $X_1, X_2, ..., X_n$ represent a random sample of size n. The only assumption in this test is that the sample is random. Denote the null hypothesis by

$$H_0 : F(x) = H(x) \text{ for all } x \tag{2.11}$$

where $F(x)$ is the unknown population distribution function, and $H(x)$ is the hypothesized distribution function with all parameters specified. $H(x)$ may be continuous, discrete, or a mixture of two types. Let $S_n(x)$ represent the empirical distribution function,

$$S_n(x) = \frac{1}{n}(\text{the number of } X_i's \text{ which are} \leq x) \tag{2.12}$$

Then the two sided test statistic $D = \sup_x |H(x) - S_n(x)|$. If the test statistic is greater than the critical value, null hypothesis is rejected. Calculation of critical values for discrete distributions is described in Conover (1972). However, Arnold and Emerson (2010) showed that when the sample size is larger than 30, critical value cannot be calculated using the method described by Conover (1972) and suggested to use the approximate critical values for the discrete distributions. Arnold and Emerson (2010) modified the $ks.test()$ package in R for the discrete distributions. This modified package is used to get the approximate P-values for the simulated data.

Data	D	P Value
$l5\ \tilde{g}0$	0.11	0.178
$l5\ \tilde{g}1$	0.06	0.864
$l5\ \tilde{g}2$	0.08	0.544
$l5\ \tilde{g}3$	0.04	0.997
$l5\ \tilde{g}4$	0.08	0.544
$l5\ \tilde{g}5$	0.06	0.864
$l5\ \tilde{g}6$	0.05	0.964
$l5\ \tilde{g}7$	0.06	0.864
$l5\ \tilde{g}8$	0.09	0.393

Table 2.2: Kolmogorov Smirnov test results for 100 random RNA sequences with 1000 nucleotide bases.

$$H_0 \ : \ \text{Data follows a Poisson Distribution}$$

$$H_1 \ : \ \text{Data does not follow a Poisson Distribution}$$

From Table 2.2, it can be seen that the null hypothesis is not rejected at 5% level when $l = 5$ and $\tilde{g} = 0,1,\ldots,$ and 8. This test was done for all 54 parameter combinations considering l=3,...,8 and $\tilde{g}$=0,...,8. It was found that, Poisson approximation is valid for number of inversions from all 54 parameter combinations.

2.4.2 Cramèr-von Mises Test

Test Procedure

The Cramèr-von Mises criterion is a criterion used for judging the goodness of fit of a probability distribution $F^*(x)$ compared to a given empirical distribution function F_n,

or for comparing two empirical distributions. This criterion for testing that a sample $X_1, X_2, \ldots, X_n$, has been drawn from a specified continuous distribution $F(X)$ is

$$W^2 = \int_{-\infty}^{\infty} \left[F_N(x) - F(x)\right]^2 dF(x) \tag{2.13}$$

where $F_N(x)$ is the empirical distribution function of the sample; that is $F_N(x) = k/N$ if exactly k observations are less than or equal to x, $(k = 0, 1, \ldots, N)$ (Anderson, 1962).

Spinelli and Stephens (1997) modified the test statistics of Cramèr-von Mises test for the Poisson distributions and the corresponding critical values. Suppose the random variable X can take integer values w_j from 0 to ∞. Let p_j be the probability of observing value j. Suppose further that n independent observations of X are given, labeled $x_1, x_2, \ldots, x_n$. Let o_j be the observed number of observations equal to j, and $np_j = e_j$ be the expected number of values j, $Z_j = \sum_{i=0}^{j} (o_i - e_i)$ and $H_j = \sum_{i=0}^{j} p_i$ where $i, j = 0, 1, 2, \ldots$. The Cramèr-von Mises statistics W^2, A^2, and W_m^2 are then defined by

$$W^2 = n^{-1} \sum_{j=0}^{\infty} Z_j^2 p_j \tag{2.14}$$

$$A^2 = n^{-1} \sum_{j=0}^{\infty} \frac{Z_j^2 p_j}{H_j (1 - H_j)} \tag{2.15}$$

$$W_m^2 = n^{-1} \sum_{j=0}^{\infty} Z_j^2 \tag{2.16}$$

In general, when a tested distribution has unknown parameters, they should be estimated from he sample by an efficient method such as maximum likelihood (Spinelli & Stephens, 1997). Let $\hat{p}_j$ be the estimated probability of observing value j. The $\hat{e}_j$, $\hat{Z}_j$,$\hat{H}_j$ are calculated replacing p_j with $\hat{p}_j$. For practical calculations, the sums above must be finite and the distribution must be truncated. It is proposed by Spinelli and Stephens (1997) that this can be done by choosing the truncation point on right tail, at cell M_u so that $O_j = 0$ and $\hat{p}_j < 10^{-3}/n$ for $j > M_u$. Here O_j is the observed number of observations equal to j and $\hat{p}_j$ is the estimated probability of observing value j. They suggested that, truncation at the lower end can be made at cell M_l such that $O_j = 0$ and $\hat{p}_j < 10^{-3}/n$ for $j < M_l$. Inclusion

of further terms will not significantly change the value of a typical test statistic (Spinelli & Stephens, 1997).The new formulae become:

$$W^2 = n^{-1} \sum_{j=M_l}^{M_u} Z_j^2 p_j \tag{2.17}$$

$$A^2 = n^{-1} \sum_{j=M_l}^{M_u} \frac{Z_j^2 p_j}{H_j \left(1 - H_j\right)} \tag{2.18}$$

$$W_m^2 = n^{-1} \sum_{j=M_l}^{M_u} Z_j^2 \tag{2.19}$$

The algorithm is coded in an R script (see Appendix B) and the 3 test statistics, W^2, A^2, and W_m^2 are obtained for 9 simulated data sets. These 9 data sets were obtained by calculating the number of inversions for inversion parameter combinations $l5\ \tilde{g}1$, $l5\ \tilde{g}2$, ..., $l5\ \tilde{g}8$. Results are as in table 2.3.

Data	W^2	A^2	W_m^2	Simulated Mean	W^2 Table	A^2 Table	W_m^2 Table
$l5\ \tilde{g}0$	0.007	0.005	0.037	0.890	0.203	1.191	0.624
$l5\ \tilde{g}1$	0.017	0.058	0.114	1.870	0.182	1.151	0.881
$l5\ \tilde{g}2$	0.008	0.018	0.090	2.430	0.182	1.151	0.881
$l5\ \tilde{g}3$	0.074	0.127	0.515	3.070	0.182	1.151	0.881
$l5\ \tilde{g}4$	0.092	0.092	0.634	3.820	0.182	1.151	0.881
$l5\ \tilde{g}5$	0.161	0.184	1.186	4.600	0.172	1.112	1.359
$l5\ \tilde{g}6$	0.179	0.234	1.166	5.150	0.172	1.112	1.359
$l5\ \tilde{g}7$	0.053	0.059	0.442	5.870	0.172	1.112	1.359
$l5\ \tilde{g}8$	0.045	0.093	0.550	6.640	0.172	1.112	1.359

Table 2.3: Cramèr-von Mises test results

The unknown Poisson mean is estimated from the simulated data. Since the test statistics

are less than the corresponding table values suggested by Spinelli and Stephens (1997), except for W^2 in $l5\ \tilde{g}6$ case, we can say that, the null hypothesis that the data are a random sample from a Poisson distribution is not rejected. The outlying observation in the $l5\tilde{g}6$ case may be due to a random chance in the simulation process. Therefore, this test also verifies that the distribution of inversions can be approximated by the Poisson distribution.

This test was carried out using all 54 parameter combinations and only 5 parameter combinations had one or two test statistic values which exceed the table values at 5% significance level. Further, non of the parameter combinations had all three test statistic values greater than corresponding table values. Therefore, it can be concluded that the number of inversions follow a Poisson distribution for the 54 parameter combinations considered.

2.4.3 Chi-Square Goodness of Fit Test

The chi-square test is defined for the hypothesis:

H_0 : Data follows a Poisson Distribution

H_1 : Data does not follow a Poisson Distribution

For the Chi-square Goodness-of-fit computation, the data is divided into k bins and the test statistic is defined as

$$\chi^2 = \sum_{i=1}^{k} \left(O_i - E_i\right)^2 / E_i \tag{2.20}$$

where O_i is the observed frequency for bin i and E_i is the expected frequency for bin i under null hypothesis.

The test statistic follows, approximately, a chi-square distribution with $(k - c)$ degrees of freedom where k is the number of non-empty cells and c is the number of estimated

parameters. Therefore, the hypothesis that the data is from a population with the specified distribution is rejected if $\chi^2 > \chi^2_{\alpha,k-c}$ at significance level of α. The Chi-square goodness-of-fit test is done using the R functions, $goodfit()$ and results are as in table 2.4.

Data	Pearson‘s χ^2	Df	$P(\geq X^2)$
$l5\ \tilde{g}0$	0.266	2	0.875
$l5\ \tilde{g}1$	2.242	5	0.815
$l5\ \tilde{g}2$	3.136	6	0.791
$l5\ \tilde{g}3$	7.703	8	0.463
$l5\ \tilde{g}4$	9.025	10	0.530
$l5\ \tilde{g}5$	17.505	11	0.094
$l5\ \tilde{g}6$	13.607	12	0.326
$l5\ \tilde{g}7$	7.524	13	0.873
$l5\ \tilde{g}8$	7.916	14	0.894

Table 2.4: Chisquare goodness of fit test results

From table 2.4 it can be seen that the null hypothesis is not rejected at 5% level for $l = 5$ with $\tilde{g} = 0,\ldots,8$. We further carried out this test for all 54 parameter combinations as well and found out that, Poisson approximation for the number of inversions is a good approximation. Therefore, it can be concluded that the distribution of the number of inversions can be approximated by a Poisson distribution.

Chapter 3

Algorithms for Segmenting RNA Sequences

3.1 Why do we need to segment the RNA sequences?

Predicting RNA secondary structure is a complicated process, requiring so much computer time and memory that often makes it impractical or impossible to perform any detailed predictions for sequences only a few hundred bases long. Yet, there exists RNA molecules (e.g., in some viral genomes) of biological interest that contain over a thousand bases. Taufer et al. (2008) developed an approach to overcome computer resource limitations by the grid computing technology. They have successfully used this technology for protein structure predictions (Taufer, An, Kerstens, & Brooks, 2006) already and are adapting the grid computing approach for RNA predictions. The idea is to segment a long RNA sequence into smaller chunks and send them to different computers on the grid for individual predictions. Then, the individual predictions are assembled to give the prediction for the original molecule.

The challenge for the grid computing approach is the loss of information incurred by the segmentation process. While there are different RNA secondary structure prediction algorithms available, all of them utilize the nucleotide sequence information of the entire RNA molecule. If the molecule is cut into chunks to be processed independently, all sequence information outside of a chunk will not be used for its secondary structure prediction. Furthermore, if, during the segmentation process, a cut is placed between the two stems of

an inversion contained in a secondary structure element, it will become impossible for this element to be predicted. As a result the overall prediction will be less accurate. In this chapter, I attempt to investigate the effects of 2 different strategies for segmentation on the overall prediction accuracy of the entire molecule. Both of these strategies involve first identifying those areas in the RNA sequence with high concentration of inversions and then keeping each area in the same chunk.

3.2 Identification of the Locations of the Inversions

In order to locate the areas with high concentration of inversions, we first have to locate where the inversions are. An adapted version of the "Palindrome" program in the EMBOSS package was used to identify the inversions of the RNA sequence and extract the positions of their occurrences in the sequences. We had to modify the EMBOSS palindrome program in order to fit our purpose because the original program works for DNA but not for RNA sequences. Also we need to extract the information about the positions and counts of the inversions for our computations. So we used an adapted version which was written in Java. We called this new version the "Inversion" program and its source code is included in Appendix A.

```
AUAUGAUUUAAAAGAGUUUGUACUGCGAAGUGGCACACCUAAGCAGUGUACGGUGACU
CCUGCCCAAUCGGUAGCCCUCUGAGACCAAGCACAGCGGUCAGUAUAAUAACACAACA
UCGUUCGACCGUUGGGCCGUGGUACCAUUCACUCUGCAGCGAUCAUACCGUUUCGUG
CUAUAGAGGAGAGGCCUAUUGCGGUCA
```

Figure 3.1: Input for the *Palindrome* program.

A text file containing the RNA sequence can be specified as the input to the program. Figure 3.1 is a typical input file. When an RNA sequence is fed to the *Inversion* program, with the required parameters (minimum stem length and the maximum gap size of the inversions), it identifies the positions of the inversions of the sequence and output two text

```
Palindromes of:
AUAUGAUUUAAAAGAGUUUGUACUGCGAAGUGGCACACCUAAGCAGUGUACGGUGACUCCUGCCCAAUCGG
UAGCCCUCUGAGACCAAGCACAGCGGUCAGUAUAAUAACACAACAUCGUUCGACCGUUGGGCCGUGGUACC
AUUCACUCUGCAGCGAUCAUACCGUUUCGUGCUAUAGAGGAGAGGCCUAUUGCGGUCA
Sequence length is: 200
Start at position: 1
End at position: 200
Minimum length of Palindromes is: 4
Maximum length of Palindromes is: 100
Maximum gap between elements is: 0
Number of mismatches allowed in Palindrome: 0
Number of Palindromes found: 2


Palindromes:
Palindrome 0:
135      GUGGU      139
         |||||
144      UACCA      140

Palindrome 1:
191      AUUG      194
         ||||
198      UGGC      195
```

Figure 3.2: An example output from the *Palindrome* program.

files. The first output file is in the same form as that of the EMBOSS palindrome program (see Figure 3.2), and the second output file is shown in Figure 3.3. The first row gives the length of the sequence. From the second row on, starting positions and the ending positions of the inversions are given by the first and the second columns of the output respectively. An example output from the inversion program was given in Figure 3.3.

200	200
135	144
191	198

Figure 3.3: An output from the *Inversion* program.

3.3 The Score Based Excursions

In a variety of bioinformatics applications, the problem calls for identifying high concentration of a certain property in nucleotide sequences. For example, in Chew, Leung, and Choi (2007), the authors look for regions that are unusually rich in the nucleotides A and T (thymine) in DNA sequences in order to help identify replication origins in DNA viral genomes. I follow their approach, called *excursions*, in this study, but with interest in the inversions in RNA sequences instead. The excursion method requires assigning to each nucleotide a positive score if it is part of an inversion (including the stems and gap), and a negative score if it does not. Then we go through the entire nucleotide sequence accumulating the scores to form excursions, as detailed below.

To facilitate the excursion analysis, I wrote a parsing program to generate a binary sequence with the same length as original sequence. If a nucleotide base is included in an inversion, then that nucleotide base was given a value 1 and 0 otherwise. Therefore, the entire RNA sequence was converted into a binary sequence, based on the inversion distribution.

. . G C G A U U G C C **G U C A** G G C A A U A C U . .
. . 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 . .

As Chew et al. (2007) suggested, nucleotide bases of interest were given a score of w and others were given a score of s. That is, the 1's in our binary sequence were given w and 0's were given s, which will be specified below. The RNA sequence was considered as a realization of a sequence of i.i.d. random variables, $X_1, X_2, \ldots, X_n$, where n is the length of the RNA sequence, taking values in $\{s, w\}$. If the i^{th} base is a 1, then $X_i = w$ and otherwise $X_i = s$. We let $Pr\,(X_i = s) = p$ and $Pr\,(X_i = w) = 1\text{-}p$ (denoted by q). The parameter, p, is naturally estimated by the percentage of bases not contained in any inversions in the RNA sequence, i.e., percentage of zeros of the binary sequence.

An additional constraint was imposed on the choice of s and w: the expected score per base $\mu = ps + qw$ should be kept negative. This condition prevents favoring long segments to be high scoring segments (Karlin, Dembo, & Kawabata, 1990). As in Chew et al. (2007), we set w=1 and $\mu = -0.5$ and the score of s was found by:

$$s = \left\lfloor \frac{\mu - qw}{p} \right\rfloor \tag{3.1}$$

where $\lfloor . \rfloor$ denotes the integer floor function.

Then the cumulative scores were calculated and since interest is in positive scores, negative scores were reset to zero. The *excursion score* E_i at position i, was defined recursively as:

$$E_0 = 0, E_i = max\left\{E_{i-1} + X_i, 0\right\}, for 1 \leq i \leq n. \tag{3.2}$$

An *excursion* starts at a point i where E_i is zero and ends at $j \geq i$ where E_j is the very next zero. The score then stays at zero until it first becomes positive again for the start of the next excursion. The *value* of an excursion is defined to be the peak score during the course of that particular excursion (Chew et al., 2007). A high excursion value indicates a high concentration of inversions in that region.

A plot of the excursion scores along the nucleotide positions of the RNA sequence offers a good visualization of the variation of inversion concentration along the sequence and can serve as a guide for choosing the cut-points for the segmentation process. Figure 3.4 shows the excursion plot of *Nodamura virus RNA1* sequence with minimum stem length 3 and maximum gap size 5.

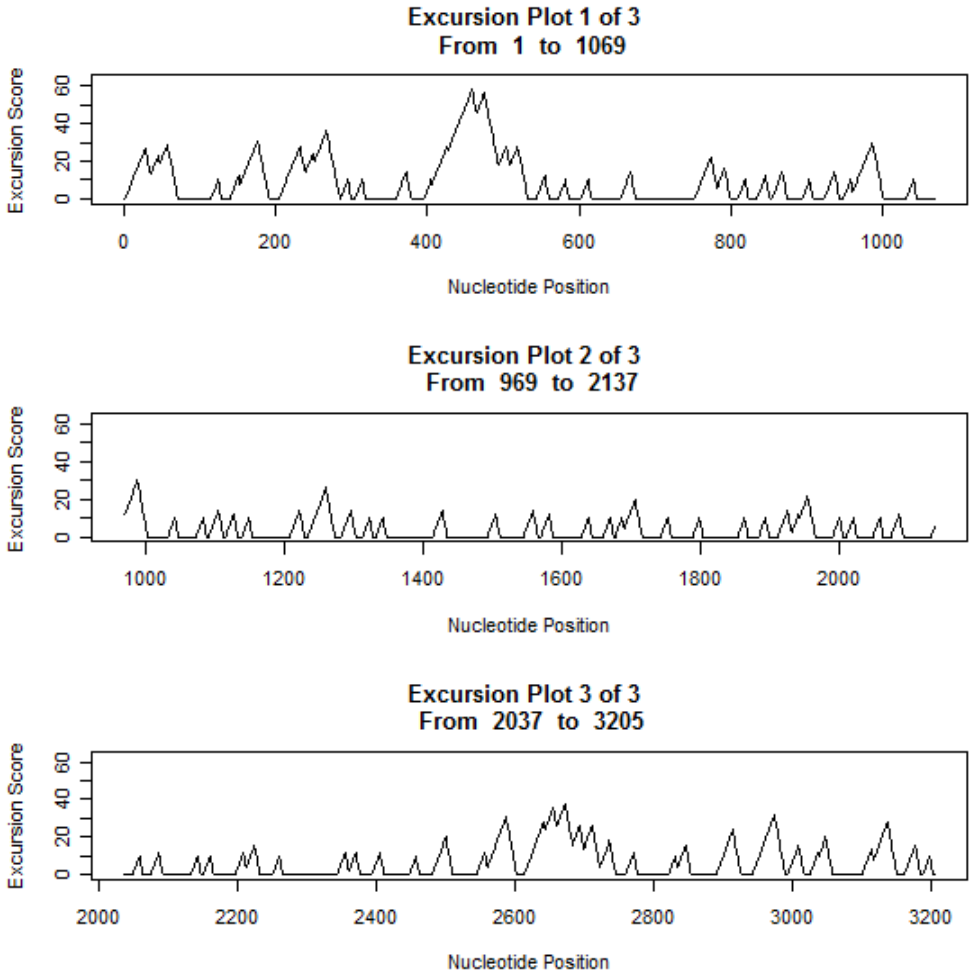


Figure 3.4: An excursion plot for Nodamura virus RNA 1 sequence with $l = 3$ and $\tilde{g} = 5$.

3.4 Algorithms for RNA Segmentation

As different computer programs for RNA structure predictions have different computer memory and time requirements, their limits on the length of the RNA sequence that can be processed also varies. For the purpose of investigating the effects of segmentation on the prediction accuracy, I used a set of RNA data sequences with known secondary structures and lengths ranging from 125 to about 570. The maximum chunk size was chosen to be 100 so that several chunks can be generated from each data sequence for testing.

After the excursion plot is obtained, peaks of the plot are identified. Then the location of the bottom of the peak, that is, the last nucleotide base which has the zero cumulative score right before any peak, is identified. After that, the length of the peak, that is, the location difference between a peak and its peak bottom, is calculated. Note that since we are only dealing with ≤ 100 base chunks, peak lengths greater than 100 have to be flagged. Their information stored and they will be considered separately. Figure 3.5 shows the peaks, peak bottoms and the peak lengths.

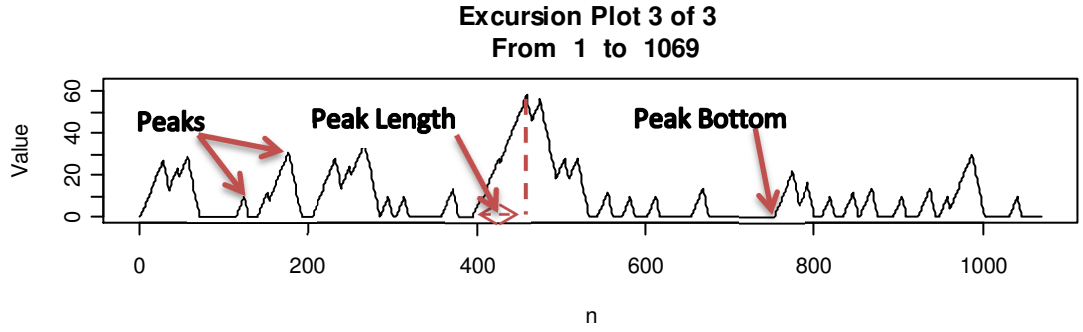


Figure 3.5: Peaks, peak bottoms, and peak lengths

Then peaks are sorted in decreasing order based on their excursion scores. We have used two methods for the segmentation process. A flow chart for the algorithm is shown in Figure 3.6.

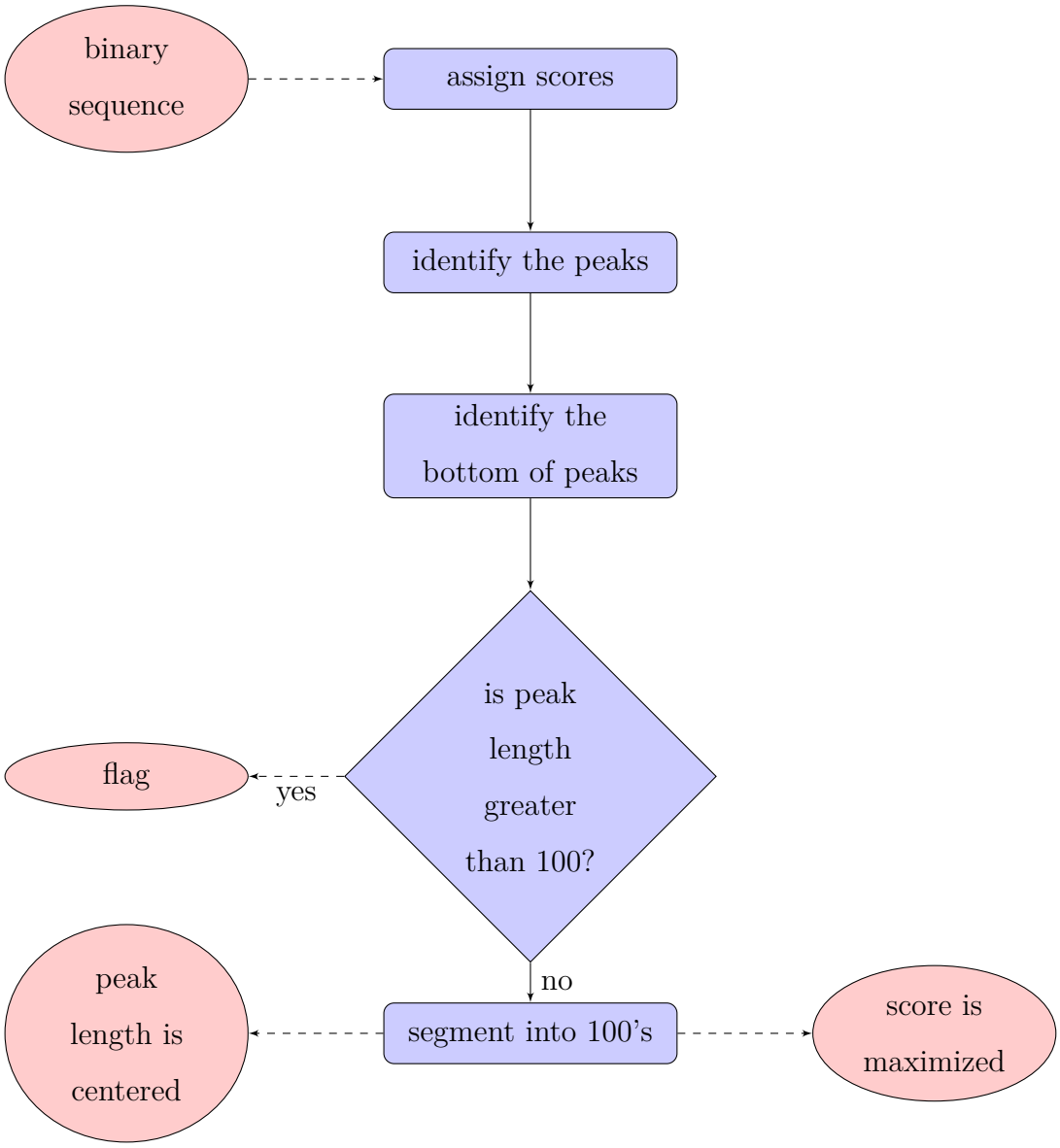


Figure 3.6: Flow Chart for the algorithm

3.4.1 Peak Length Centralizing Method (Centered Method)

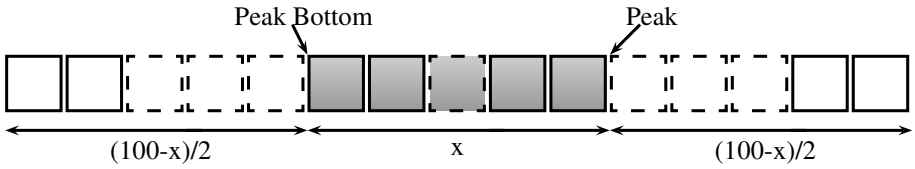


Figure 3.7: Peak length centralizing method, x = peak length

After peaks are sorted in decreasing order, the peak with the highest cumulative score is considered first. Then the cut points for the chunks are decided in a way where nucleotide bases in the peak length positions are centered in the chunk. If the chunks cover all the 1's in the sequence, then stop. Otherwise, repeat this process for the next peak with the next highest cumulative score, and so on. The algorithm stops either when all the peaks are exhausted or when all the inversion regions of the sequence (i.e all 1's in the binary sequence) are included in the chunks, which ever occurs first. Overlapping chunks are adjusted so that any nucleotide base is captured by only one chunk with priority is given to the peak with the higher excursion score.

3.4.2 Score Maximizing Method (Optimized method)

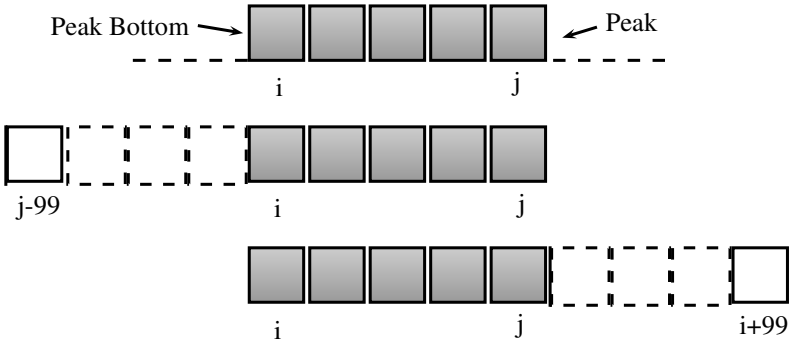


Figure 3.8: Score maximizing method

In this method, when deciding the cut points, rather than centralizing the peak length posi-

tions in the chunks, the segment of 100 nucleotide bases containing the peak and giving the highest cumulative scores are chosen. For example, consider a peak with peak length from nucleotide base i to j. Then the chunk from $(j-99)^{th}$ nucleotide base to j^{th} nucleotide base is considered and the sum of the scores of the nucleotide bases of the chunk is calculated.

This process is done for all the 100 nucleotide base chunks from $(j-99)^{th}$ nucleotide base to $(i+99)^{th}$ nucleotide base. Then the chunk with the maximum score from the above discussed set of chunks is identified. This process is repeated until all the peak lengths identified earlier. To overcome the overlapping problem, the starting position and the ending position of each chunk are adjusted after being compared to the previous chunks and to the peak lengths. This score maximizing method makes sure that peak length positions are included in the chunks, but not necessarily in the center of the chunk. The algorithms stops either when all the peaks are utilized or when all the inversion regions of the sequence are exhausted, whichever occurs first.

3.4.3 Regular segmentation method (Regular method)

A sequence can also be segmented regularly into 100 nucleotide base chunks, considering $1-100, 201-300$ and so on until all the nucleotide bases in the sequence are exhausted. This is the most convenient method of segmentation, which can be used as a reference method and we call this the *regular* method. Obviously, rising peaks were often cut by this method. Therefore, the secondary information loss is higher. Intuitively, one expects that both the *centered* and *optimized* methods will perform better in providing more accurate predictions of the overall structure of the entire RNA molecule. This leads to the question of how we should define good quantitative measures to evaluate and compare various segmentation methods.

3.5 An Illustration of the Algorithms

Our algorithms will be tested using real RNA sequences from the Rfam (Institute, 2010) database. Rfam is a collection of multiple sequence alignments and covariance models covering many common non-coding RNA families. The longest 50 sequences without the pseudoknots of the Rfam data set were selected. Names and the lengths of those sequences were shown in Table 3.1. These RNA sequences in the Rfam database have the actual secondary structures.

```
> RF00209_A.bpseq
UACGAGGUUAGUUCAUUCUCGUAUACACGAUUGGACAAAUCAAAAUUAUAAUUUGGUUCAGGGCCUCCCUCCA
GCGACGGCCGAACUGGGCUAGCCAUGCCCAUAGUAGGACUAGCAAAACGGAGGGACUAGCCAUAGUGGCGAGC
UCCCUGGGUGGUCUAAGUCCUGAGUACAGGACAGUCGUCAGUAGUUCGACGUGAGCAGAAGCCCACCUCGAGA
UGCUACGUGGACGAGGGCAUGCCCAAGACACACCUUAACCCUAGCGGGGGUCGCUAGGGUGAAAUCACGCCAC
GUGAUGGGAGUACGACCUGAUAGGGCGCCGCAGAGGCCCACUAUUAGGCUAGUAUAAAAAUCUCUGCUGUACA
UGGCACAUGGAGUU
((((((:::::::::))))))::::::::::::::::::::::::::::::::::::::::((((:::))))::::
::::::::::::::((((:::::::))))):::::::::::::::::::::::::::((((((:(((((((((:((
((((::(((((((((::((((((::::))))))::(((((:::((:((((:(((:::::::::::)))):))))::
::))::::::)))))((((::))))::))))::))))::::(((((((((((:::)))))))))))::::((((:::)
))))::::::))):)):(((::::)))::::::::::::))))))::::)))))))::::::::::::::::::((
((:::))))::::::
```

Sequence 1: RF00209_A sequence with its secondary structure from Rfam database.

A typical sequence from the Rfam database is shown in Sequence 1, with its actual secondary structure. Nucleotide bases that are involved with a stem loop are represented by a "(" or a ")" signs and a matching () pair represents two paired bases. Nucleotide bases

that are not paired are represented by a : (colon). To illustrate the use of two algorithms, the 379 base RNA sequence $RF00209\_A$, from the Rfam database is used in this section. The excursion plot of the sequence with minimum stem length 3 and maximum gap-size 5 is shown in figure 3.9.

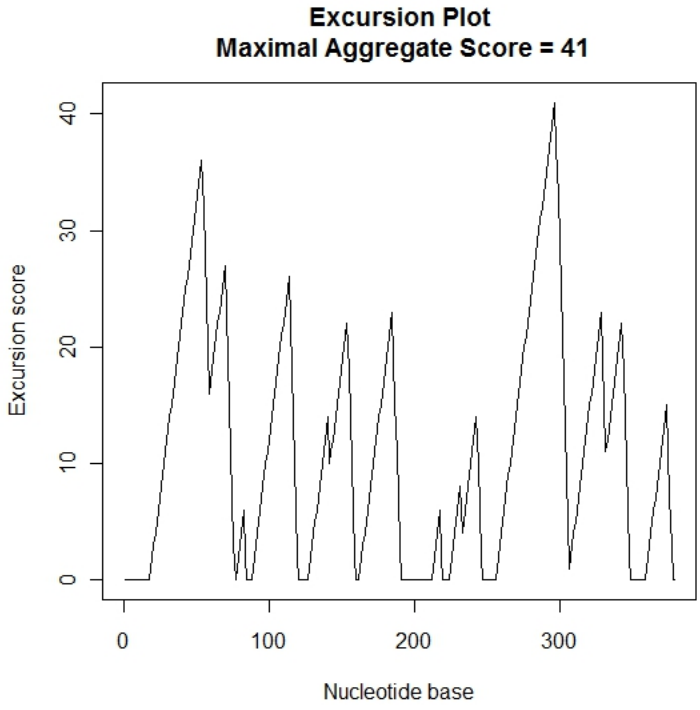


Figure 3.9: Excursion plot of $RF00209\_A$ with l=3 and $\tilde{g}$=5.

The algorithms were coded in R and the output from the program is shown in figure 3.10. The *centered* method segmented the sequence into 6 segments. These six segments cover the entire sequence. On the other hand, the *optimized* method gave 5 segments. These 5 segments used only 96.3% of the sequences. Note that the inversions always lie on the ascending portions of the peaks of the excursion plots. Our objective is to segment the RNA sequence into chunks with minimum information loss. Therefore, ascending stretches of peaks should not be cut into two parts or those situations should be avoided as much as

possible. Starting position of the segment is denoted by *Lower* and the ending position of the sequence is denoted by *Upper*.

```
[1] "Length of the sequence"
[1] 379
[1] "Centered Method"
     peakbottom peak peakvalues Lower Upper
[1,]         19    70         36     1   100
[2,]         89   114         26   101   150
[3,]        163   185         23   151   223
[4,]        225   243         14   224   255
[5,]        257   343         41   256   349
[6,]        360   374         15   350   379
[1] "Optimized Method"
     peakbottom peak peakvalues Lower Upper
[1,]         19    70         36    15   114
[2,]        128   154         22   115   159
[3,]        163   185         23   160   256
[4,]        257   343         41   257   356
[5,]        360   374         15   357   379
[1] "Centered Method_coverage"
[1] 100
[1] "number_of_segments_Centered Method"
[1] 6
[1] "Optimized Method_coverage"
[1] 96.30607
[1] "number_of_segments_Optimized Method"
[1] 5
```

Figure 3.10: Output from the R program, giving locations of the chunks resulting from the two segmentation methods, percentages of sequence covered, and number of chunks.

Segments generated from the *centered* method were shown in the excursion plot in Figure 3.11. Starting and the ending positions of the segments were indicated using red vertical lines. It can be seen that some of the ascending stretches of the peaks were cut by the segmenting process. Therefore, some of the information about the inversions was lost. This will reduce the accuracy of the predictions of the RNA secondary structures.

Segments from the *optimized* method were shown in Figure 3.12. It can be seen that all the rising peaks were avoided. Therefore, the information loss from the *optimized* method is expected to be less than the *centered* method for this sequence. Also, the *optimized*

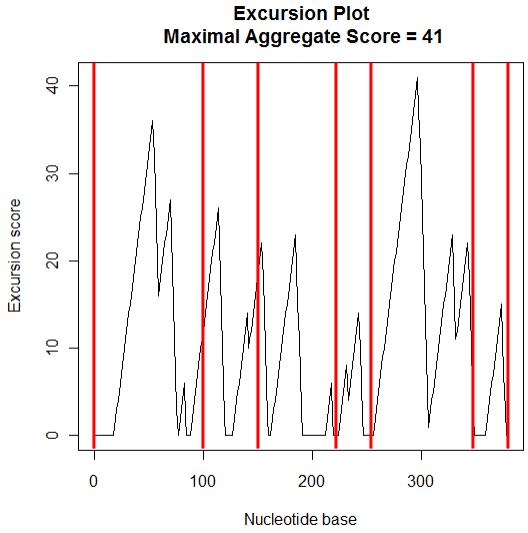


Figure 3.11: Segments from the centered method.

method used only 96.3% of the nucleotide bases in the sequence. In other words, nucleotide bases which do not have any information about the inversions were not included in the segments. Therefore, the wastage of computing resources is relatively minimized in the *optimized* method.

Segments from the *regular* method for the *RF*00209, with minimum stem length 3 and maximum gap size 5, were shown in figure 3.13.

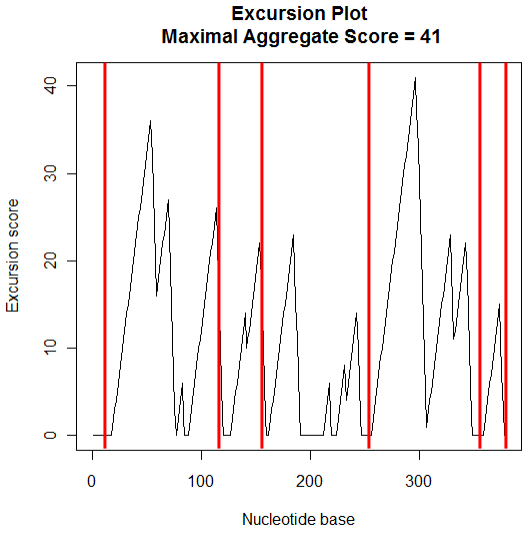


Figure 3.12: Segments from the optimized method

3.6 Evaluation of Segmentation Methods

The objective of the segmentation process is to cut large RNA sequences into smaller chunks so that separate predictions of the individual chunks can be combined to give a prediction for the whole sequence with minimal loss of accuracy. To evaluate how well different segmentation methods perform towards this goal, we propose the following quantitative measures:

1. *Consistency*: This is defined as percentage of agreement between the predicted structure of the whole sequence and the structure obtained by assembling the predicted structures of the individual chunks.

2. *Retained Accuracy*: This is defined as the ratio AC/AW, where AC is the accuracy (i.e. the percentage agreement with the known real structure) when the prediction is assembled from the chunks, and AW is the accuracy when the prediction is done on the whole sequence. The ratio will be converted to a percentage.

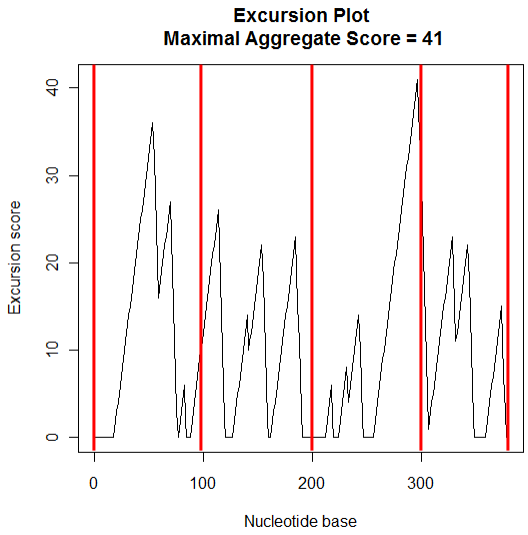


Figure 3.13: Segments from the regular method

It is expected that both of theses measures, which are related but not identical, will help us evaluate which segmentation algorithm perform better. The first measure indicates how consistent the predictions are before and after segmentation. The second measure indicates how much of the secondary structure prediction accuracy is retained after segmentation. These measures can be applied to any choice of segmentation method and secondary structure prediction algorithm. Both measures require the calculation of a percentage agreement between two possible secondary structures of the same sequence to be assessed.

I have implemented the code to compute consistency and retained accuracy values with the plan to evaluate the different segmentation methods. For a full evaluation, more time is needed to develop the program to correctly assess the percentage agreement between two possible secondary structures. However, just to verify that the codes for these evaluation measures are working as expected, I have carried out some preliminary calculations using a convenient (but inappropriate) character-by-character comparison of the two strings of "(","), and ":" between the 2 structures to assess their percentage agreement. If (,) or : is matched in both sequences at the same position, then a value of 1 is given and if it is

not then a 0. Then the sum of the 1's were divided by the length of the sequence to obtain the consistency percentage.

```
Structure 1=((((((::::(((((::(((((:::::::(::::((:::::((((((:::::::((:::))
            :::::
Structure 2=(((((((((::(((::(((((:((::::)):)))))):::::((((((::::((((((:::
            ::)))
Agreement  =11111100010111010111100011110010000011111111111111110001100100
            11000
```

Sequence 2: The character-by-character comparison of two structures.

Note that a character-by-character comparison of the bracket views of 2 structure is not a completely correct assessment because it does not take the pairing information into account. For example, consider the two strings ((((::)))) and :((::))(:). If they are compared character by character, they are 50% in agreement because their characters at positions 2,3,5,7,10 are the same. However, the first sequence indicates pairings of (1,10), (2,9), (3,8), (4,7) and unpaired bases 5 and 6. The second sequence indicates pairings of (2,7), (3,6), and (8,10) and unpaired bases 1,4,5,9. So, the only thing in common between the two structures is the unpaired base 5. So, considered as secondary structures, they should only be only be 10% in agreement.

The consistency and retained accuracy were calculated for each of the sequences in our dataset (see Table 3.1) with the centered, optimized, and regular segmentation methods, and secondary structure prediction algorithms RG (Reeder & Giegerich, 2004) and UNAFOLD (Markham & Zuker, 2008). In the segmentation process with the centered and optimized methods, I used minimum stem-length $l = 3, 4, \ldots, 8$ and maximum gap-sizes $\tilde{g}$=0,1,...,8, giving a total of 54 (l,$\tilde{g}$) combinations. Some of these values are shown in Table 3.2 and Table 3.3. The mean and standard deviation over the entire dataset of RNA sequences were then calculated. Figure 3.14 and Figure 3.15 display the 95% confidence

bands for the 4 prediction algorithm and segmentation method combinations with $l = 3$ and 4. From all the confidence bands I have examined, no significant difference in the performance of the different segmentation methods has been detected. However, I expect that both the centered and optimized method would perform better than the regular method when a more appropriate agreement assessment becomes available.

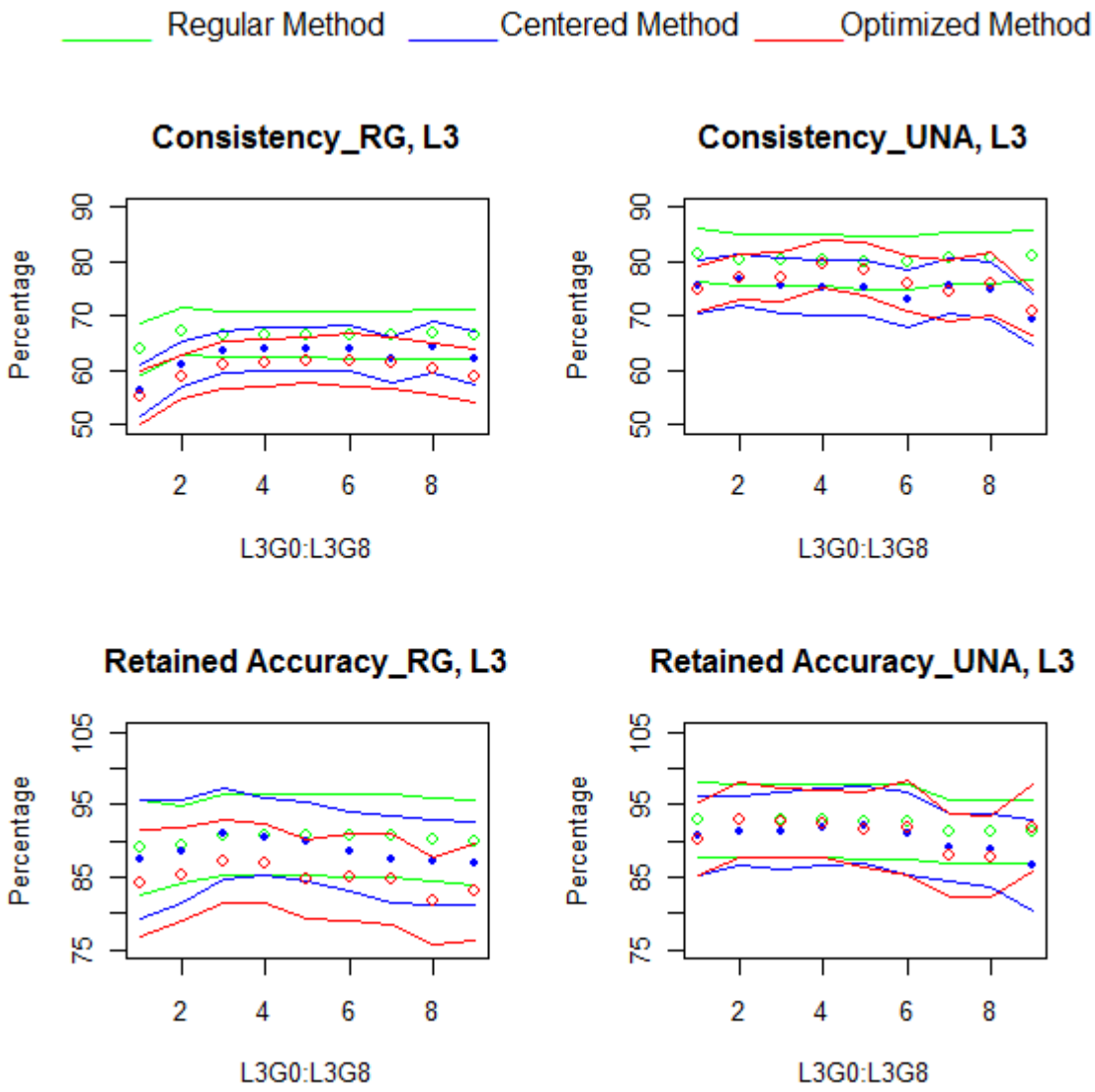


Figure 3.14: Summary of the consistency ratios for L=3

Sequence Name	Length	Sequence Name	Length
RF00488_B	568	RF00232_A	170
RF00177_A	482	RF00171_A	168
RF00210_A	462	RF00387_A	168
RF00209_A	379	RF00003_B	161
RF00011_B	374	RF00234_B	160
RF00023_B	363	RF00102_B	159
RF00028_B	344	RF00389_B	158
RF00040_B	338	RF00050_A	157
RF00036_A	337	RF00025_A	152
RF00100_A	330	RF00433_B	152
RF00009_A	320	RF00225_B	151
RF00461_B	309	RF00002_B	150
RF00017_B	305	RF00484_A	149
RF00030_A	297	RF00444_B	147
RF00503_A	293	RF00004_A	145
RF00231_A	275	RF00492_B	144
RF00193_A	273	RF00506_B	144
RF00230_B	246	RF00015_B	141
RF00012_B	219	RF00290_A	140
RF00045_B	213	RF00264_B	134
RF00487_B	211	RF00373_A	133
RF00013_A	183	RF00437_B	131
RF00252_B	182	RF00007_B	129
RF00168_B	179	RF00182_A	129
RF00391_A	171	RF00463_A	127

Table 3.1: Names and the lengths of the 50 longest sequences without pseudoknots of the Rfam database

RF00488_B	Centered(%)	Optimized(%)	Regular(%)
L3G0	48.94	48.06	38.91
L3G1	50.70	51.41	38.91
L3G2	51.94	50.00	38.91
L3G3	51.41	57.22	38.91
L3G4	53.87	54.22	38.91
L3G5	52.29	54.05	38.91
L3G6	49.30	61.27	38.91
L3G7	53.52	59.15	38.91
L3G8	45.77	54.93	38.91

Table 3.2: Consistency comparison of methods using PknotsRG prediction software for RF00488_B.

RF00488_B	Centered(%)	Optimized(%)	Regular(%)
L3G0	117.60	109.14	94.36
L3G1	107.38	97.18	94.36
L3G2	106.68	107.04	94.36
L3G3	110.20	94.00	94.36
L3G4	108.08	104.92	94.36
L3G5	104.22	113.72	94.36
L3G6	104.22	111.26	94.36
L3G7	114.08	116.18	94.36
L3G8	100.34	109.86	94.36

Table 3.3: Retained Accuracy of methods using PknotsRG prediction software for RF00488_B.

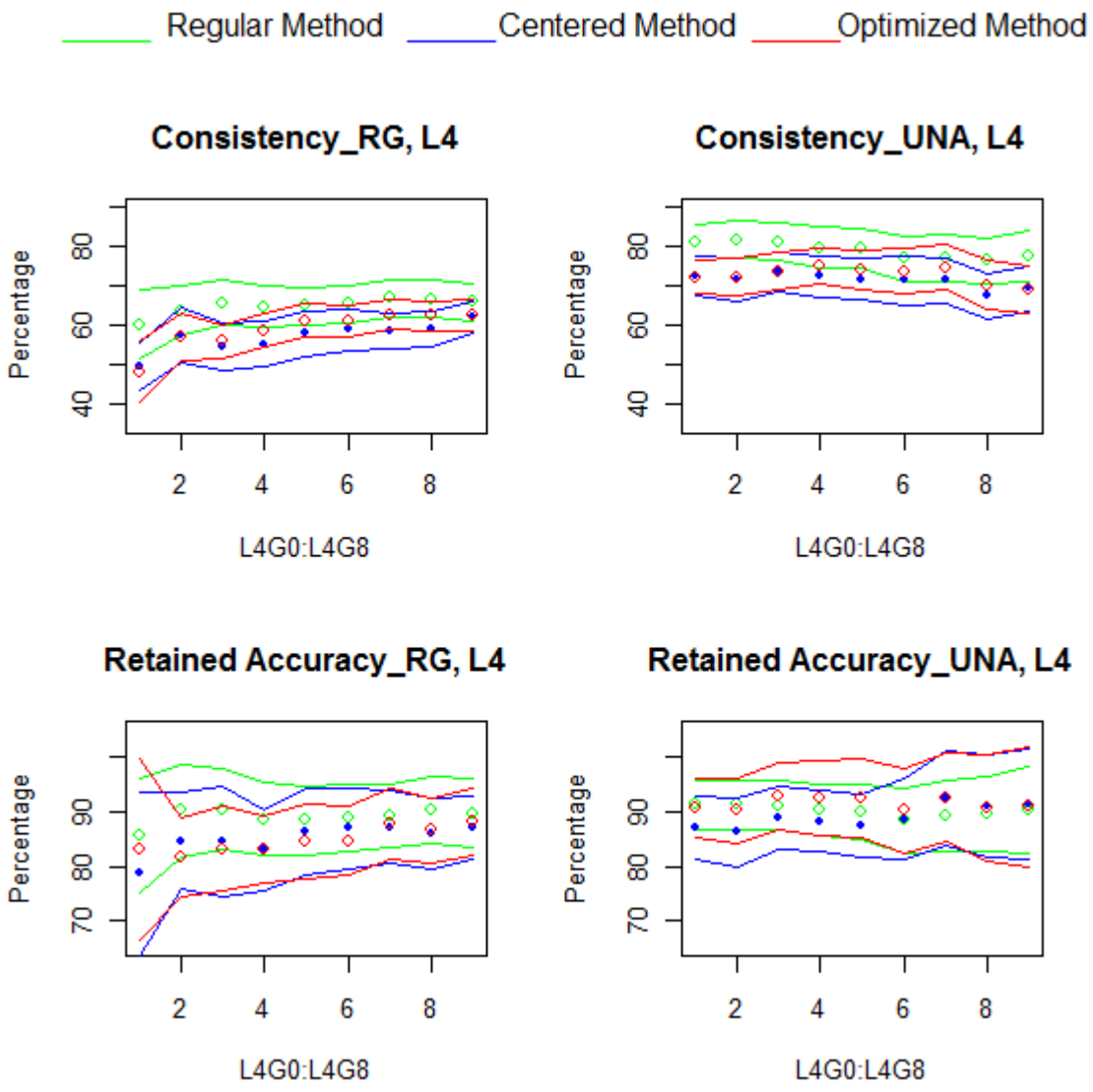


Figure 3.15: Summary of the consistency ratios for L=4

Chapter 4

General Discussions, Conclusion and Future Work

4.1 General Discussions

In this study, distributional properties of inversions are studied and number of inversions with a particular minimum stem length and a maximum gap size was found to have approximately a Poisson distribution. Leung, Choi, Xia, and Chen (2005) proved that under suitable conditions, the count of inversions with gap 0 has an approximate Poisson distribution. My simulation results demonstrated that the number of inversions with minimum stem length ≥ 3 and maximum gap sizes 0 up to 8 can be approximated by a Poisson distribution. Poisson approximation holds for rare events. When the maximum gap size is increased or the minimum stem length is increased, the number of inversions increases. So, one would expect that the Poisson approximation will become less accurate. It will be interesting to see the behavior of the Poisson approximation with varying stem lengths and gap sizes to see when the Poisson approximation ceases to hold.

Through out this study, the nucleotide sequence was considered as a realization of sequence of i.i.d. random variables. The recursive formula obtained for expected number of inversions works under this assumption. In most practical scenarios, nucleotide bases are neither independent nor identically distributed. Occurrence of a certain nucleotide base might depend on neighboring bases. For this reason, Markov models are quite commonly need in nucleotide sequence analysis studies. In this thesis, I used the i.i.d. model for mathematical

tractability, but it would be worthwhile to generalize the analysis to Markov models for a longer term project.

4.2 Conclusion

There are two main findings in this study. A recursive formula for expected number of inversions with minimum stem length 3 or more and maximum gap size from 0 to 8 in a random nucleotide sequence was obtained. Furthermore, distribution of number of inversions can be approximated by the Poisson distribution for the parameter combinations examined.

Character by character comparison method does not consider the actual secondary structure of the sequence. It is very important to compare the prediction structures obtained using the segmentation algorithms, by a better structure similarity measure. A method that considers the actual base pairing in secondary structures, should be employed to evaluate the performances of the segmentation algorithms.

4.3 Future Work

In this study, only the $G-C$ and $A-U$ pairings of the RNA sequences were considered. However, one can also take the wobble pairing (i.e. $G-U$ pairing), which is possible in RNA but occurs less frequently in general, into account and generalize the recursive formula for the expected number of inversions. The distributional properties for the new inversion count allowing for wobble pairing will again be an interesting topic for investigation.

In the segmentation algorithms, we only considered the inversions which are less than or equal to 100 nucleotide bases. It is important to have an idea about the distribution of the span of inversions, i.e. the distribution of the length of the inversions, including the two stems and the gap in between. If we know how likely to have a large inversion in a sequence, then we can design the segmentation methods accordingly. Therefore, as an extension of this work, one can investigate the distributional properties of inversion spans. This will help us develop segmentation algorithms to effectively utilize the available computing resources to deal with long inversions which in this study were flagged and left to manual handling. The inversion span distribution will help us design better segmentation strategies to improve overall prediction accuracies with minimal manual work.

Bibliography

Anderson, T. (1962). On the distribution of the two sample Cramer-von Mises criterion. *Annals of Mathematical Statistics*, *33*, 1148-1159.

Arnold, T., & Emerson, J. (2010). Nonparametric goodness-of-fit tests for discrete null distributions. *The R Journal*.

Chew, D., Leung, M.-Y., & Choi, K. P. (2007). At excursion: a new approach to predict replication origins in viral genomes by locating at-rich regions. *BMC Bioinformatics*.

Conover. (1972). A kolmogorov goodness-of-fit test for discontinuous distributions. *Journal of the American Statistical Association*, *67*.

Faller, M. (1999). *Emboss-palindrome @ONLINE.* Available from `http://emboss.bioinformatics.nl/cgi-bin/emboss/palindrome`

Institute, H. H. M. (2010). *Rfam 10.0 @ONLINE.* Available from `http://rfam.janelia.org/index.html`

Karlin, S., Dembo, A., & Kawabata, T. (1990). Statistical composition of high-scoring segments from molecular sequences. *The Annals of Statistics*, *18*(2), 571–581. Available from `http://dx.doi.org/10.2307/2242124`

Leung, M. Y., Choi, K. P., Xia, A., & Chen, L. H. (2005). Nonrandom clusters of palindromes in herpesvirus genomes. *Journal of Computational Biology*, *12*, 331-354.

Markham, N. R., & Zuker, M. (2008). Unafold: software for nucleic acid folding and hybriziation. *Bioinformatics*, *II*.

Reeder, J., & Giegerich, R. (2004). Design, implmentation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics. *BMC Bioinformatics*, *5:104*.

Rivas, E., & Eddy, S. (1999). A dynamic programming algorithm for RNA structure prediction including pseudoknots. *Journal of Molecular Biology*.

Spinelli, J., & Stephens, M. (1997). Cramer-von mises test of fit for poisson distribution.

Canadian Journal of Statistics, *25*, 257-268.

Taufer, M., An, C., Kerstens, A., & Brooks, C. (2006, aug.). Predictor@home: A "protein structure prediction supercomputer' based on global computing. *Parallel and Distributed Systems, IEEE Transactions on*, *17*(8), 786 -796.

Taufer, M., Leung, M.-Y., Solorio, T., Licon, A., Mireles, D., Araiza, R., et al. (2008). Rnavlab: A virtual laboratory for studying rna secondary structures based on grid computing technology. *ScienceDirect*, *34*, 661-680.

Appendix A

Java Codes for Identifying and Counting the Inversions

```
//should be saved as PalFinder.java

import java.io.*;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.PrintWriter;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.Scanner;

/**
 * This program is designed to find palindromes in sequences of RNA or DNA
 * In this context is palindrome of a sequence is its reverse complement
 *
 * Input is a raw ascii file
 * Output the palindromes to a file you name
 *
 * @author Peter Kelley
 * Adapted from program Palindrome.c as in the EMBOSS suite
 *
 */

public class PalFinder {
```

```
private boolean rNA = true;
private boolean gU = true;

public static void main(String[] args) {

    int minLen, maxLen, maxGap, beginPos, endPos, maxmismatches;

    String seqstr;
    int count, gap,length;
    maxGap = 8;// maximum gap
    length=100;//length of the sequence

    for(int f=0;f<9;f++){
   int[] a = new int[length+1];
    maxGap = f;
    int begin, end, mismatches, mismatchAtEnd, istart, iend, ic, ir;
    boolean alln;  /* TRUE if all of palindrome is N's */
    boolean printToFile = false;
    //LinkedList<Palindrome> palList = new LinkedList<Palindrome>();
    PalFinder palObj = new PalFinder();
    boolean found = false, overlap = false;
    for(int d=1;d<length+1;d++){
      LinkedList<Palindrome> palList = new LinkedList<Palindrome>();
    String pathName = "random"+d+".txt";

    BufferedReader inputStream = null;
try {
 inputStream = new BufferedReader(new FileReader(pathName));
} catch (FileNotFoundException e) {
 System.out.println("Could not open: "+pathName+ " Please enter a file that exists.");
 System.exit(0);
}
Scanner fileScanner = new Scanner(inputStream);

String sequence = palObj.getSeq(fileScanner);
    String rna = "Y";
    if (rna.matches("[Tt]rue|[Yy]es|[Yy]")){
     if(sequence.indexOf('T')!=-1||sequence.indexOf('t')!=-1){
      System.out.println("Converted T's to U's...");
      sequence = sequence.replace('T', 'U');
      sequence = sequence.replace('t', 'u');
     }
```

```
 String gu = "N";
 if (!gu.matches("[Tt]rue|[Yy]es|[Yy]")){
  palObj.setGU(false);
 }
}else{
 if(sequence.indexOf('U')!=-1||sequence.indexOf('u')!=-1){
  System.out.println("You cannot process the sequence as DNA if it contains U's.");
  System.exit(1);
 }
 System.out.println("The sequence will be processed as DNA");
 palObj.setRNA(false);
}
if (minLen<1){
 System.out.println("Minimum less than 1. Reset to 1.");
 minLen=1;
}else if(minLen>sequence.length()/2){
 minLen = sequence.length()/2;
}
maxLen = 1000;
if (maxLen<1){
 System.out.println("Maximum less than minimum. Reset to "+minLen);
 maxLen = minLen;
}else if(maxLen>sequence.length()/2){
 maxLen = sequence.length()/2;
}
if (maxGap<0){
 System.out.println("Maximum gap must be 0 or greater. Reset to 0.");
 maxGap=0;
}else if(maxGap>sequence.length()){
 maxGap = sequence.length();
}
maxmismatches = 0;
if (maxmismatches<0){
 System.out.println("Minimum less than 0. Reset to 0.");
 maxmismatches=0;
}
String overlapStr = "N";

if (overlapStr.matches("[Tt]rue|[Yy]es|[Yy]")){
 overlap = true;
}else{
 overlap = false;
```

```
    }

    String printToFilestr = "N";
    PrintWriter outFile = null;
    // start and end for file printing
    beginPos = 1;
    endPos = sequence.length();
    if (printToFilestr.matches("[Tt]rue|[Yy]es|[Yy]")){
     printToFile = true;
     /* write header to file */

  try {
   myOutFile = new FileWriter(outfileName);
  } catch (IOException e) {
   System.out.println("Could not open: "+outfileName+" for output.");
   System.exit(0);
  }
  outFile = new PrintWriter(myOutFile,true);

  outFile.format("\n%-8d ", sequence.length());
  outFile.format("%-8d ",sequence.length());
}
 //   System.out.println("\nFinding Palindromes...");
   //set up variables for loop
 seqstr = sequence;
 begin = beginPos-1;
 end = endPos-1;

 /* loop to look for inverted repeats */
 for(int current = begin; current < end; current++){

  iend = current + 2*(maxLen) + maxGap; // 2 because it is a repeat?
     if(iend > end){
      iend = end;        // if end of this palindrome goes past end, make actual end the end
     }
     istart = current + minLen;    // set up start, palindrome must start here at least...

     //work backwards looking for palindromes, only go back to the min length of a palindrome
     for(int rev = iend; rev > istart; rev--)
     {
      //reset the mismatches count
      count = 0;
```

```
    mismatches = 0;
    mismatchAtEnd = 0;
    alln = true;
    //use these to search for palindrome, they include current up to minlen
    ic = current;
    ir = rev;

    if(palObj.baseComp(seqstr.charAt(ic), seqstr.charAt(ir))){
     while(mismatches <= maxmismatches && ic < ir)
     {
      if(palObj.baseComp(seqstr.charAt(ic++), seqstr.charAt(ir--)))
      {
       mismatchAtEnd = 0;
       if(seqstr.charAt(ic-1) != 'n'){
        alln = false;
       }
      }
      else
      {
       mismatches++;
       mismatchAtEnd++;
      }
      count++;
     }
    }
    count -= mismatchAtEnd; //if it ends with a mismatch then remove it from the count
gap = rev - current - count - count + 1; //calculate gap between forward and reverse

// if it is a legal palindrome add it
if(count >= minLen && gap <= maxGap && !alln)
{
    /* create new object to hold palindrome data */
    Palindrome ppal = new Palindrome(current,(current+count),rev,(rev-count));

    /*
    ** if it is the first palindrome find then save it as start
    **  of palindrome list
    * check if it is compatible with the over palindromes, overlapping....
    */
    if (palList.size() == 0){
     palList.add(ppal);
    }else{
```

```
        ListIterator<Palindrome> itr = palList.listIterator();
        found = false; // see if this is a valid palindrome to add
        while(itr.hasNext()){
         Palindrome pnext = (Palindrome) itr.next();
         if(overlap && palObj.palindrome_AInB(ppal, pnext)) //yes, report overlaps and AinB
         {
          // if we do want to report overlaps
          // only report when A is NOT IN B, A IS NOT A SUBSET OF B
          found = true;
          break;
         }
       //no, do not report overlaps and AoverB
          if(!overlap && palObj.palindrome_AOverB(ppal, pnext))
          {
           if(palObj.palindrome_Longer(ppal, pnext)){
            palObj.palindrome_Swap(ppal, pnext);
           }
           found = true;
           break;
          }
        }

        /* if new palindrome add to end of list */
     if(!found){
      palList.add(ppal);
     }
    }
   }
     }
}

ListIterator<Palindrome> itr;
a[d]=palList.size();
System.out.println(a[d]);
if(printToFile==true){
 /* Print out the palindromes to outfile */
 itr = palList.listIterator();
 int palCount=0;
 while(itr.hasNext())
 {
  palObj.palindrome_Print(outFile, seqstr, (Palindrome) itr.next(), maxLen);
  palCount++;
```

```
 }
}

//System.out.println("\nSaving Palindromes...");
itr = palList.listIterator();
while(itr.hasNext())
{
 Palindrome currPal = itr.next();
 currPal.setForwardEnd(currPal.getForwardEnd()-1);
 currPal.setRevEnd(currPal.getRevEnd()+1);
 if(currPal.getForwardEnd() - currPal.getForwardStart() > maxLen){
  itr.remove();
 }
}



}

FileOutputStream fout;

try
{
    // Open an output stream
    fout = new FileOutputStream ("countL"+minLen+"G"+maxGap+".txt");


    for (int i5 = 1; i5 < length+1; i5++) {  // i indexes each element successively.
      //sum += ar[i];

    // Print a line of text
    new PrintStream(fout).println (a[i5]);
    }
    // Close our output stream
    fout.close();
}
// Catches any error conditions
catch (IOException e)
{
 System.err.println ("Unable to write to file");
 System.exit(-1);
}
}
```

```
}

private String getSeq(Scanner fileScanner) {
 String sequence = "";
 String header = fileScanner.nextLine();
 char first = header.charAt(0);

 if(first==';'){
  // This is a .seq file
  String next = fileScanner.nextLine();
  System.out.println("Note for .seq if fails:");
  System.out.println("Name of sequence must be on the line directly after any comments...");
  while(Character.toString(next.charAt(0)).matches("[;]")){
   next = fileScanner.nextLine();
  }
  next = fileScanner.next();
  sequence = sequence.concat(next);
  while(fileScanner.hasNext()){
   sequence = sequence.concat(fileScanner.next());
  }
  return sequence.substring(0,sequence.length()-1);

 }else if(first=='>'){
  // This is a fasta file
  while(fileScanner.hasNext()){
   sequence = sequence.concat(fileScanner.next());
  }
  return sequence;
 }else{
  // This is a raw file
  sequence = sequence.concat(header);
  while(fileScanner.hasNext()){
   sequence = sequence.concat(fileScanner.next());
  }
  return sequence;
 }

}


private void palindrome_Print(PrintWriter outFile, String seq,
  Palindrome pal, int maxLen) {
```

```
    if(pal.getForwardEnd() - pal.getForwardStart() <= maxLen){
     outFile.format("\n%-8d ", (pal.getForwardStart()+1));
     int j=pal.getRevStart();
     for(int i = pal.getForwardStart(); i < pal.getForwardEnd(); i++){
      if(baseComp(seq.charAt(i), seq.charAt( j--))){
       outFile.print("|");
      }else{
       outFile.print(" ");
      }
     }
     for(int i = pal.getRevStart(); i > pal.getRevEnd(); i--){
      outFile.format("%c",seq.charAt(i));
     }
    }

}

private void palindrome_Swap(Palindrome a, Palindrome b) {
 b.setForwardStart(a.getForwardStart());
    b.setForwardEnd(a.getForwardEnd());
    b.setRevStart(a.getRevStart());
    b.setRevEnd(a.getRevEnd());

}

private boolean palindrome_Longer(Palindrome a, Palindrome b) {
 if((a.getForwardEnd() - a.getForwardStart()) >
       (b.getForwardEnd() - b.getForwardStart())){
  return true;
 }
 return false;
}

private boolean palindrome_Over(int astart, int aend, int bstart,
       int bend) {
 if(astart >= bstart && astart <= bend){
  return true;
 }
 if(bstart >= astart && bstart <= aend){
  return true;
```

```
 }

 return false;
}

private boolean palindrome_AOverB(Palindrome a, Palindrome b) {
  if(palindrome_Over(a.getForwardStart(), a.getForwardEnd(),
          b.getForwardStart(), b.getForwardEnd()) &&
        palindrome_Over(a.getRevEnd(), a.getRevStart(),
          b.getRevEnd(), b.getRevStart())){
 return true;
}
 return false;
}

private boolean palindrome_AInB(Palindrome a, Palindrome b) {
 if((a.getForwardStart() >= b.getForwardStart()) && (a.getForwardEnd() <=b.getForwardEnd())){
  if((a.getRevStart() <= b.getRevStart()) && (a.getRevEnd() >= b.getRevEnd())){
   return true;
  }
 }
 return false;
}

private boolean baseComp(char c, char d) {
 if(rNA){
  if(c==baseCompRNA(d)){
   return true;
  }
  if(gU){
   if (Character.toString(c).matches("[GUgu]")&&c==baseCompRNAalt(d)){
    return true;
   }
  }

  return false;

 }else{
  if(c==baseCompDNA(d)){
   return true;
  }else{
   return false;
```

```
  }
 }
}

private char baseCompRNAalt(char d) {
 // these were adjustsed to only work with RNA
 // cannot make them work for both as A would be ambiguous
 // possibly just add A to fwd and U to rev

 String fwd="GUgu";
    String rev="UGug";

    int baseInd = fwd.indexOf(d);
    if(baseInd!=-1)
    {
     return rev.charAt(baseInd);
    }

    return d; // but a does equal a, can't return this...
}

private char baseCompRNA(char c) {

 // these were adjustsed to only work with RNA
 // cannot make them work for both as A would be ambiguous
 // possibly just add A to fwd and U to rev

 String fwd="AUCGRYWSMKBDHVNXaucgrywsmkbdhvnx";
    String rev="UAGCYRWSKMVHDBNXuagcyrwskmvhdbnx";

    int baseInd = fwd.indexOf(c);
    if(baseInd!=-1)
    {
     return rev.charAt(baseInd);
    }

    return c;
}

private char baseCompDNA(char c) {

 // These strings are left unchanged from the original palindrome program
```

```
  // They do have RNA bases but will not use them if not present
  // won't fully work with RNA
  // possibly just add A to fwd and U to rev
  String fwd="ACGTURYWSMKBDHVNXacgturywsmkbdhvnx";
      String rev="TGCAAYRWSKMVHDBNXtgcaayrwskmvhdbnx";

      int baseInd = fwd.indexOf(c);
      if(baseInd!=-1)
      {
       return rev.charAt(baseInd);
      }

      return c;
 }

 public boolean isRNA() {
  return rNA;
 }
 public void setRNA(boolean rna) {
  rNA = rna;
 }

 public boolean isGU() {
  return gU;
 }
 public void setGU(boolean gu) {
  gU = gu;
 }

}
```

Required Java Class-Palindrome.java

```
//Should be saved as Palindrome.java
/**
 * @author Peter Kelley
 * Adapted from program Palindrome.c as in the EMBOSS suite
 *
 * This class holds the data related to one palindrome that is
 * found in the sequence
 *
 * It also has getters and setters for all the data fields
 *
 * the data members hold the location of the palindrome
 * Note: It is divided into two sequences, first and second halves
 */

public class Palindrome implements java.io.Serializable{

 private static final long serialVersionUID = 592685737506127943 7L;
 // Note: These are all 0 indexed, add 1 for printing
 private int forwardStart;
 private int forwardEnd; //end is larger than start
 private int revStart;
 private int revEnd; //end is smaller than start

 private int score;
 private int palNum; // indexed by 0, i am sure
 private boolean include;
 public Palindrome(int fstart, int fend, int rstart, int rend) {
  setForwardStart(fstart);
  setForwardEnd(fend);
  setRevStart(rstart);
  setRevEnd(rend);

 }

 public void setForwardStart(int forwardStart) {
  this.forwardStart = forwardStart;
 }

 public int getForwardStart() {
  return forwardStart;
```

```
}

public void setForwardEnd(int forwardEnd) {
 this.forwardEnd = forwardEnd;
}

public int getForwardEnd() {
 return forwardEnd;
}

public void setRevStart(int revStart) {
 this.revStart = revStart;
}

public int getRevStart() {
 return revStart;
}

public void setRevEnd(int revEnd) {
 this.revEnd = revEnd;
}

public int getRevEnd() {
 return revEnd;
}

public void setScore(int score) {
 this.score = score;
}

public int getScore() {
 return score;
}

public void setPalNum(int palNum) {
 this.palNum = palNum;
}

public int getPalNum() {
 return palNum;
}
```

```
 public void setInclude(boolean include) {
  this.include = include;
 }

 public boolean isInclude() {
  return include;
 }
}
```

Appendix B

R Script for Cramèr-von Mises Test

```
result=matrix(0,nrow=9,ncol=4,dimnames=list(c("L5G0","L5G1","L5G2","L5G3","L5G4","L5G5","L5G6",
        "L5G7","L5G8"),c("W2","A2","Wm2","Mean")))

for(ii in 0:8){
#Read the text files which has the number of inversions in random RNA sequences
x=read.table(paste("C:/Users/Sameera/Documents/My Dropbox/Pen Drive/Simulation
        /100/countL5G",ii,".txt",sep=""))
x=x[,1]
n=length(x)
min=min(x)
max=max(x)
x=sort(x)
data=0
data[1:(max-min+1)]=min:max
y=0
d=0
for (i in 1:(max-min+1)){
        for (j in 1:n){
                if (data[i]==x[j]){
                d=d+1
                }
        }
data[i]=d
d=0
}
data
sum(data)
p=min:max
oj=data
lamda=mean(x)
pjhat=dpois(p,lamda)
ej=n*pjhat
k=length(oj)
```

```
l=matrix(0,nrow=length(oj),ncol=2)
l[,1]=pjhat
l[,2]=oj
e=1/(1000*n)
#-------------------------------------------------------------
Mu=0
Ml=0
for(kk in 2:length(oj)){
if((l[kk,1]<e)&(l[kk,2]==0))
Mu=which((l[kk,1]<e)&(l[kk,2]==0))-1
else {Mu=length(oj)}
}
for(kk2 in 1:(Mu-1)){
if((l[kk2,1]<e)&(l[kk2,2]==0))
Ml=which((l[kk2,1]<e)&(l[kk2,2]==0))+1
else{Ml=1}
}

Sj=0
for (j in Ml:Mu){
Sj[j+1]=oj[j]+Sj[j]
}
Sj=Sj[-1]
Tj=0
for (j in Ml:Mu){
Tj[j+1]=ej[j]+Tj[j]
}
Tj=Tj[-1]
Zj=Sj-Tj
Hj=Tj/n
W2=sum(Zj*Zj*pjhat)/n
Wm2=sum(Zj*Zj)/n
A2=(sum((Zj*Zj*pjhat)/Hj*(1-Hj)))/n
result[ii+1,1]=W2
result[ii+1,2]=A2
result[ii+1,3]=Wm2
result[ii+1,4]=lamda
}

result
```

Appendix C

R Script for Segmenting RNA Sequences

```
pathS="RF00009_A.bpseqL"  #name of the file which has the inversion position information
fileName=NA
lll=1

for (k in 3:8){
        for (kk in 0:8){
                seqName=paste(pathS,k,"G",kk,sep="")
                cut=NA
                cut2=NA
                newcord=NA
                stretch=NA
                segName=NA
                dd=NA
                dd2=NA
                cut0=NA
                segName3=substring(seqName,1,(nchar(seqName)-10))
                readPath=paste("C:/Users/Sameera/Documents/My Dropbox/Pen Drive/Simulation/Simulate
                        /",seqName,".txt",sep="")

cc=100  # for the bonus scheme
#read the output from the Palindrome Java Program
position=read.table(readPath, header = FALSE,na=TRUE)

#-------For the Regular Method

long=position[1,1]
cut0=matrix(0,nrow=ceiling(long/100),ncol=2)
hundred=100
one=1
```

```
#Segments the entire sequence into hundreds
for (j in 1:(ceiling(long/100))){
        cut0[j,1]=one
        cut0[j,2]=hundred
        one=one+100
        hundred=one+99
        hundred=min(hundred,long)
}
#------------------------------

if(nrow(position)>2){

        m=nrow(position)-1
        alpha=0.001
        fill=seq(0,0,length=position[1,1])
        for (i in 1:m){
                fill[position[i+1,1]:position[i+1,2]]=1
        }

#-----Score based method
#-----Excursion scores are stored in "E"

indicator=fill
n=length(indicator)
q1=sum(indicator)/n
p1=1-q1
w=1
mu=-0.5
s=floor((mu-q1*w)/p1)
p1=round(p1,4)
q1=round(q1,4)
label=rbind(c("p","q","w","s"),c(p1,q1,w,s))
E=array(0, dim=c(n,1))
E[0]=0
for (i in 1:n) {
if (indicator[i]==1){
E[i]=max(E[i-1]+w,0)
}
else
{
E[i]=max(E[i-1]+s,0)
}
```

```
}
Mn=max(E)

if (s>1)
        {
                zero=rep(0,s-2)
                z=c(-1,q1,zero,p1)
        }
if (s<0)
        {
                zero=rep(0,abs(s)-1)
                z=c(p1,zero,-1,q1)
        }

if (s!=0 | s!=1)
        {
                ff=polyroot(z)
                dd=Re(ff)
        }

for (pp in 1:length(dd))
        {
                if ((dd[pp]!=1) & (dd[pp]>0))
                y=dd[pp]
        }

#---Solve for lamda star--------

lamdastar=log(y)
kstar=(exp(-lamdastar)-exp(-2*lamdastar))*(s*p1*exp(lamdastar*s)+q1*exp(lamdastar))

#-------------------------------------------------
x1=log(log((1-0.05))/-kstar)/(-lamdastar)
sigLevel1=(log(n)/lamdastar)+x1
sigline1=rep(sigLevel1,n)
#-------------------------------------------------
x2=log(log((1-0.01))/-kstar)/(-lamdastar)
sigLevel2=(log(n)/lamdastar)+x2
sigline2=rep(sigLevel2,n)
#-------------------------------------------------
x3=log(log((1-0.001))/-kstar)/(-lamdastar)
sigLevel3=(log(n)/lamdastar)+x3
```

```
sigline3=rep(sigLevel3,n)

#-------------------sepaerate into several plots
if (n>1000)
{
num=ceiling(n/3)
par(mfrow=c(3,1))
j=1
overlay=0
ymax=Mn+5
for (i in 1:3){
plot(((j-overlay):(j+num)),E[(j-overlay):(j+num)],type="l",ylab="Value",xlab="n",lwd=1,
        ylim=c(0, ymax),main=paste("Excursion Plot",i ,"of 3","\n Maximal Aggregate Score =",
          Mn,"\n From ",j-overlay," to ",j+num))
plot(((j-overlay):(j+num)),E[(j-overlay):(j+num)],type="l",ylab="Value",xlab="Index",lwd=1,
         ,ylim=c(0, ymax),main=paste("Excursion Plot",i ,"of 3","\n From ",j-overlay," to ",j+num))
lines(sigline1,col="red")
lines(sigline2,col="blue")
lines(sigline3,col="green")
j=j+num
overlay=100

}
legend("topright",c('5%','1%','0.1%'),lty=c(1,2,3), horiz=TRUE)
legend("topright",c('5%'),inset = .05,lty=c(1),ncol = 1,lw=c(2.5),col=c("Red"))

}
if (n<=1000)
{
par(mfrow=c(1,1))
plot(E,type="l",ylab="Value",xlab="n",lwd=1
        ,main=paste("Excursion Plot\n Maximal Aggregate Score =",Mn))
lines(sigline1,col="red")
lines(sigline2,col="blue")
lines(sigline3,col="green")
}
legend("topright",c('5%'),inset = .05,lty=c(1),ncol = 1,lw=c(2.5),col=c("Red"))
label1=rbind(c("Lamda Star","K Star","X"),c(lamdastar,kstar,x1))


#----Identifying Peaks using the excursion scores----------------
peak=0
```

```
jp=1

for (ip in 1: length(E)){
        if( E[ip]<E[ip+1] & E[ip+1]>E[ip+2] & ip+2<=length(E)){
                peak[jp]=ip+1
                jp=jp+1
        }
}

coord1=peak[]
peakbottom=0

for (jpc in 1:length(coord1)){
        jpb=coord1[jpc]
                while (E[jpb]>0)
                {
                        jpb=jpb-1
                }
        peakbottom[jpc]=jpb+1
}

#--------------- Identify peak bottom from the other side
coord2=peak[]
peakbottom2=0

if (E[n]==0){
        for (jpc1 in 1:length(coord2)){
                jpb1=coord2[jpc1]
                        while (E[jpb1]>0)
                        {
                        jpb1=jpb1+1
                        }
                        peakbottom2[jpc1]=jpb1
                }
}

if (E[n]>0){
        for (jpc1 in 1:(length(coord2)-1)){
                jpb1=coord2[jpc1]
                        while ((E[jpb1]>0)&&(E[jpb1]<=E[n]))
                        {
                        jpb1=jpb1+1
```

```
                }
                peakbottom2[jpc1]=jpb1

                }
peakbottom2[length(coord2)]=n
}

#----------------------------
peakvalues=E[coord1[]]
coord=cbind(peakbottom,coord1)
flag1=0
coordvalue=cbind(coord,peakvalues,flag1,peakbottom2)
coordvalue[,4]=ifelse(((coord[,2]-coord[,1])>99),1,0)
flagLow=coordvalue[coordvalue[,4]!=0,1]
flagHigh=coordvalue[coordvalue[,4]!=0,2]
coordvalue=subset(coordvalue,coordvalue[,4]!=1)
library(doBy)
dd=orderBy(~-peakvalues, data=coordvalue)

if (length(dd[,1])>1){
        dd=dd[,-c(4)]
        ddBottom=dd
        dd=dd[,-c(4)]
}

if (length(dd[,1])==1){
        dd=dd[-c(4)]
        ddBottom=dd
        dd=dd[-c(4)]
}
#-------------------------remove inclusive peaks before doing anything

comon=0
ddBottom=cbind(ddBottom,comon)

for (ib in 1:(length(ddBottom[,1])-1)){
        for (ia in ib:(length(ddBottom[,1])-1)){
                if ((ddBottom[ib,1]==ddBottom[ia+1,1])&&(ddBottom[ia,2]>=ddBottom[ia+1,2]))
                {
                ddBottom[ia+1,5]=1
                }
```

```
                    if ((ddBottom[ib,1]==ddBottom[ia+1,1])&&(ddBottom[ia,2]<=ddBottom[ia+1,2]))
                    {
ddBottom[ia+1,5]=1
ddBottom[ib,3]=max(ddBottom[ia+1,3],ddBottom[ib,3])
ddBottom[ib,2]=max(ddBottom[ia+1,2],ddBottom[ib,2])
                    }
          }
}

lenOut=sum(ddBottom[,5]!=1)
ddBottom=ddBottom[ddBottom[,5]!=1,]

if (lenOut>1){#1
          ddBottom=ddBottom[,-c(5)]
          ddcheck=ddBottom
          dd=ddBottom[,-c(4)]

#-----------------------Centered Method---------------------------------------
l=50-(dd[,2]-dd[,1])/2
Lower=floor(dd[,1]-l)
Upper=floor(dd[,2]+l)-1
Upper=ifelse((Upper<100),100,Upper)
Lower=ifelse((Lower>=(length(E)-100)),(length(E)-100),Lower)
Lower=ifelse((Lower<1),1,Lower)
Upper=ifelse((Upper>length(E)),length(E),Upper)
dd1=cbind(dd,Lower)
overlap=0
keep=0
stretch=cbind(dd1,Upper,overlap,keep)

#----------------Checking for overlaps--------------------------------
dimen0=0
dimen00=0
stretch=orderBy(~-peakvalues, data=stretch)
for (iss in 1:(length(stretch[,1])-1)){
for (js in (iss+1):length(stretch[,1])){
if ((stretch[iss,1]<=stretch[js,1]) & (stretch[iss,2]>=stretch[js,2])& (stretch[js,6]!=1)){
stretch[js,6]=1
}
}
}
dimen0=sum(stretch[,6]!=1)
```

```
stretch=subset(stretch, stretch[,6] == 0)
#------------------------------

if(dimen0!=1){
        for (iss in 1:(length(stretch[,1])-1)){
                for (js in (iss+1):length(stretch[,1])){
                        if ((stretch[iss,4]<=stretch[js,1]) & (stretch[iss,5]>=stretch[js,2])&
                                        (stretch[js,6]!=1)){
                                stretch[js,6]=1
                                stretch[iss,7]=1
                        }
                }
        }
dimen00=sum(stretch[,6]!=1)
}

if ((dimen0!=0)&&(dimen00!=0))
        {
        dimen0=min(dimen0,dimen00)
        }

if ((dimen0!=0)&&(dimen00==0))
        {
        dimen0=dimen0
        }

if ((dimen0==0)&&(dimen00!=0))
        {
        dimen0=dimen00
        }

#---remove overlapped segments-------------------------

stretch=subset(stretch, (stretch[,6] == 0 & stretch[,7] == 0)|
        (stretch[,6] == 0 & stretch[,7] == 1)|(stretch[,6] == 1 & stretch[,7] == 1))

#---reverse check--------------------------------------

if (dimen0>1){
cutreverse=orderBy(~peakvalues, data=stretch)
        for (iss in 1:(length(stretch[,1])-1)){
                for (js in (iss+1):length(stretch[,1])){
```

```
                        if ((cutreverse[iss,4]<=cutreverse[js,1]) &
                                (cutreverse[iss,5]>=cutreverse[js,2]))
                                cutreverse[js,6]=1
                }
        }

dimen10=sum((cutreverse[,6] == 0 & cutreverse[,7] == 0)|(cutreverse[,6] == 0 &
                cutreverse[,7] == 1)|(cutreverse[,6] == 1 & cutreverse[,7] == 1))
cutreverse=subset(cutreverse, (cutreverse[,6] == 0 & cutreverse[,7] == 0)|
                (cutreverse[,6] == 0 & cutreverse[,7] == 1)|(cutreverse[,6] == 1 & cutreverse[,7] == 1)

if (dimen10>1){#3
cutreverse=orderBy(~-Lower, data=cutreverse)
        for (iss in 1:(length(cutreverse[,1])-1)){
                for (js in (iss+1):length(cutreverse[,1])){
                        if ((cutreverse[iss,1]<=cutreverse[js,1]) & (cutreverse[iss,2]>=cutreverse[js,2
                                cutreverse[js,6]=1
                                        if (cutreverse[iss,3]>=cutreverse[js,3]){
                                                cutreverse[iss,3]=cutreverse[iss,3]
                                        }
                                        else
                                                cutreverse[iss,3]=cutreverse[js,3]
                        }
                }
        }

dimen11=sum((cutreverse[,6] == 0 & cutreverse[,7] == 0)|
        (cutreverse[,6] == 0 & cutreverse[,7] == 1)|(cutreverse[,6] == 1 & cutreverse[,7] == 1))

if (dimen11>1){#4
cutreverse=subset(cutreverse, (cutreverse[,6] == 0 & cutreverse[,7] == 0)|
        (cutreverse[,6] == 0 & cutreverse[,7] == 1)|(cutreverse[,6] == 1 & cutreverse[,7] == 1))
        for (iss1 in 1:(length(cutreverse[,1])-1)){
                for (js1 in (iss1+1):length(cutreverse[,1])){
                        if ((cutreverse[iss1,4]<=cutreverse[js1,4]) & (cutreverse[iss1,5]
                                        >=cutreverse[js1,5])){
                                cutreverse=cutreverse[-js1,]
                        }
                        if ((cutreverse[iss1,1]<=cutreverse[js1,1]) & (cutreverse[iss1,2]
                                        >=cutreverse[js1,2])){
                                cutreverse=cutreverse[-js1,]
```

```
                        }
                }
        }
#-------------------------------------------------
cut=orderBy(~peakvalues, data=cutreverse)

for(kl2 in 1:nrow(cut)){
        for (kl in 1:nrow(cut)){
                if ((cut[kl2,5]==cut[kl,4])&&(cut[kl2,3]>=cut[kl,3])){
                        cut[kl,4]=cut[kl,4]+1
                        cut[kl,5]=cut[kl,5]+1
                }
                if ((cut[kl2,5]==cut[kl,4])&&(cut[kl2,3]<cut[kl,3])){
                        cut[kl2,5]=cut[kl2,5]-1
                        cut[kl2,4]=cut[kl2,4]-1
                }
        }
}

#--------------
cut=orderBy(~-peakvalues, data=cut)

for(kl2 in 1:nrow(cut)){
        for (kl in 1:nrow(cut)){
                if ((cut[kl2,5]==cut[kl,4])&&(cut[kl2,3]>=cut[kl,3])){
                        cut[kl,4]=cut[kl,4]+1
                        cut[kl,5]=cut[kl,5]+1
                }
                if ((cut[kl2,5]==cut[kl,4])&&(cut[kl2,3]<cut[kl,3])){
                        cut[kl2,5]=cut[kl2,5]-1
                        cut[kl2,4]=cut[kl2,4]-1
                }
        }
}

cut=orderBy(~Lower, data=cut)
numseg=length(cut[,1])
cutPL=cut[,4]
cutPU=cut[,5]
}
```

```
if (dimen0==1){
        cut=stretch[-c(6,7)]
        cutPL=cut[4]
        cutPU=cut[5]
        numseg=dimen0
}

notP1=0
notP1=cutPL[1]-1
coverP1=0
tti=1
tt=1

if (dimen0>1){
        while (coverP1<100 & tt<=length(cutPL)-1)
                {
                        if((cutPU[tt]<cutPL[tt+1]) & tt<=(length(cut[,1])-1))
                {
                        notP1=notP1+cutPL[tt+1]-cutPU[tt]-1
                        coverP1=100-(notP1/length(E))*100
                }
        else
                {
                        coverP1=100-(notP1/length(E))*100
                }
                tt=tt+1
}

if (cutPU[length(cutPU)]<length(E)){
        notP1=notP1+(length(E)-cutPU[length(cutPU)]-1)
}
coverP1=100-(notP1/length(E))*100
covN=c(rep(0,length(E)))

for (ii1 in 1:length(cutPL)){
        for (i1 in (cutPL[ii1]):(cutPU[ii1])){
                covN[i1]=covN[i1]+1
        }
}
}

if (dimen0==1){
```

```
        coverP1=(((cut[5]-cut[4])+1)/length(E))*100
}
coverP1

}#3
}#4

cutPL=cutreverse[4]
cutPU=cutreverse[5]


#-----Optimized method------------

if (lenOut>1){#2
        penalty=-1
        dpenalty=0.5
        startt=0
        lp=0
        sumscore=0
        sumcoord=0
        summax=0
        summax2=0
        lastt=0
        comodi=0
        dd2=ddBottom
        fill2=seq(0,0,length=position[1,1])
        chink=0
        newcord1=matrix(NA,ncol=6,nrow=length(dd2[,1]))
        colnames(newcord1)=c("peakbottom","coord1","peakvalues","peakbottom2","Lower","Upper")
        overlap=0
        keep=0
        main=0
        newcord1=cbind(newcord1,overlap,keep,main)
        newcord1[,1]=dd2[,1]
        newcord1[,2]=dd2[,2]
        newcord1[,3]=dd2[,3]
        newcord1[,4]=dd2[,4]
        starttt=0
        lasttt=0

#startt ------------------------------------------------------------------
for (gh in 1:length(dd2[,1]))
```

```
{
        startt1=0
        startt2=0
        startt3=0

if(newcord1[gh,9]==0){
        startt=dd2[gh,2]-99

        if(startt<1)
        {
        startt=1
        }
        if(length(na.omit(newcord1[,5]))>=1){
                for (bi in 1:length(na.omit(newcord1[,5]))){
                        mm=which(!is.na(newcord1[,5]))
                        nn=mm[bi]
                if((newcord1[nn,6]==startt)&&(startt>newcord1[nn,5])&&
                                (newcord1[nn,9]==0)&&(startt!=1))
                        {
                        startt=cbind(startt,(newcord1[nn,6]+1))
                        startt1=max(startt1)
                        }
                }
                for (bi in 1:length(na.omit(newcord1[,5]))){
                        mm=which(!is.na(newcord1[,5]))
                        nn=mm[bi]
                        if((newcord1[nn,6]>startt)&&(startt>newcord1[nn,5])&&
                                (newcord1[nn,9]==0)&&(startt!=1))
                        {
                        startt=cbind(startt,(newcord1[nn,6]+1))
                        startt=max(startt)#just changed
                        }
                }

#----------------
                for (bi in 1:length(na.omit(newcord1[,5]))){
                        mm=which(!is.na(newcord1[,5]))
                        nn=mm[bi]
                        if((newcord1[nn,6]>startt)&&(dd2[gh,2]==newcord1[nn,5])&&
                                (newcord1[nn,9]==0)&&(startt!=1))
                        {
                        startt2=cbind(startt,newcord1[nn,6])
```

```
                        startt2=max(startt2)
                        }
                }
                 startt=startt
        }

start0=startt
# is inversion included in the chunk?

lp=1
sumscore=0
sumcoord=0
sumlastt=0
#startt=143

if(startt>dd2[gh,1]){
lastt=min((startt+99),n)
for (bi in 1:length(na.omit(newcord1[,5]))){  #count only the nonmissing rows
                        mm=which(!is.na(newcord1[,5]))
                        nn=mm[bi]
                                if((newcord1[nn,5]<lastt)&&(dd2[gh,1]<=newcord1[nn,5])&&
                                        (newcord1[nn,9]==0))
                                {
                                        lastt=cbind(lastt,(newcord1[nn,5]-1))
                                        lastt=min(lastt)#im here
                                }
                                Lower=start0
                                Upper=lastt
                        }
}
#if(startt<=dd2[gh,1]){
        while (startt<=dd2[gh,1])
        {

                        fill2=seq(0,0,length=n)
                        lastt=min((startt+99),n)

#************************************************
                if(length(na.omit(newcord1[,5]))>=1){

                        for (bi in 1:length(na.omit(newcord1[,5]))){  #count only the nonmissing rows
                                mm=which(!is.na(newcord1[,5]))
```

```
                nn=mm[bi]
                        if((newcord1[nn,5]<lastt)&&(dd2[gh,1]<=newcord1[nn,5])&&
                                (newcord1[nn,9]==0))
                        {
                                lastt=cbind(lastt,newcord1[nn,5])
                                lastt=min(lastt)#im here
                        }
                }

                for (bi in 1:length(na.omit(newcord1[,5]))){  #count only the nonmissin
                mm=which(!is.na(newcord1[,5]))
                nn=mm[bi]
                        if((newcord1[nn,5]==lastt)&&(dd2[gh,2]<=newcord1[nn,5])&&
                                (newcord1[nn,9]==0))
                        {
                                lastt=cbind(lastt,newcord1[nn,5])
                                lastt=min(lastt)-1
                        }
                }
        }
        sumcoord[lp]=startt
        sumlastt[lp]=lastt

        for (hhh in 1:length(dd2[,1]))
        {
#1

                for (j in 1:length(dd2[,4])){
                if ((startt<=dd2[hhh,1])&&(lastt>=dd2[hhh,4])&&(startt<=dd2[j,1])&&
                        (lastt>=dd2[j,4])&&(hhh!=j)&&(newcord1[hhh,9]==0))
                {
                #score for the main peak
                fill2[dd2[hhh,1]:dd2[hhh,2]]=1
                fill2[(dd2[hhh,2]+1):dd2[hhh,4]]=dpenalty
                #scores for the rest of the peaks
                fill2[dd2[j,1]:dd2[j,2]]=1
                fill2[(dd2[j,2]+1):dd2[j,4]]=dpenalty
                }
        }

#7
```

```
                    for (j in 1:length(dd2[,4])){
                        if ((startt<=dd2[hhh,1])&&(lastt>=dd2[hhh,4])&&(hhh!=j)&&(startt>dd2[j,
                                &&(startt<dd2[j,2])&&(newcord1[hhh,9]==0))
                        {
                        fill2[startt:dd2[j,2]]=penalty
                        fill2[(dd2[j,2]+1):dd2[j,4]]=-0.5
                        fill2[dd2[hhh,1]:dd2[hhh,2]]=1
                        fill2[(dd2[hhh,2]+1):dd2[hhh,4]]=dpenalty
                        }
                    }
#8

                    for (j in 1:length(dd2[,4])){
                        if ((startt<=dd2[hhh,1])&&(lastt>=dd2[hhh,4])&&(hhh!=j)
                                &&(lastt>dd2[j,1])&&(lastt>=dd2[j,2])&&(newcord1[hhh,9]==0))
                        {
                        fill2[dd2[j,1]:lastt]=1
                        fill2[(dd2[j,2]+1):dd2[j,4]]=-0.5
                        fill2[dd2[hhh,1]:dd2[hhh,2]]=1
                        fill2[(dd2[hhh,2]+1):dd2[hhh,4]]=dpenalty
                        }
                    }
#9
                    for (j in 1:length(dd2[,4])){
                        if ((startt<=dd2[hhh,1])&&(lastt>=dd2[hhh,4])&&(hhh!=j)
                                &&(lastt>dd2[j,1])&&(lastt<dd2[j,2])&&(newcord1[hhh,9]==0))
                        {

                        fill2[startt:lastt]=-1
                        #fill2[(dd2[j,2]+1):dd2[j,4]]=-0.5
                        fill2[dd2[hhh,1]:dd2[hhh,2]]=1
                        fill2[(dd2[hhh,2]+1):dd2[hhh,4]]=dpenalty
                        }
                    }
#6

                    for (j in 1:length(dd2[,4])){
                        if ((startt<=dd2[hhh,1])&&(lastt>=dd2[hhh,4])&&(hhh!=j)
                                &&(startt>=dd2[j,2])&&(startt<dd2[j,4])&&(newcord1[hhh,9]==0))
                        {

                        fill2[startt:dd2[j,4]]=-0.5
```

```
                    fill2[dd2[hhh,1]:dd2[hhh,2]]=1
                    fill2[(dd2[hhh,2]+1):dd2[hhh,4]]=dpenalty
                    }
                }

#2
                if ((startt<=dd2[hhh,1])&&(lastt>=dd2[hhh,4])&&(newcord1[hhh,9]==0))
                    {

                    fill2[dd2[hhh,1]:dd2[hhh,2]]=1
                    fill2[(dd2[hhh,2]+1):dd2[hhh,4]]=dpenalty
                    }
#3
                if ((startt<=dd2[hhh,1])&&(lastt>=dd2[hhh,2])&&(lastt<dd2[hhh,4])
                &&(newcord1[hhh,9]==0))
                    {
                    fill2[dd2[hhh,1]:dd2[hhh,2]]=1
                    }
#4
                if ((startt>dd2[hhh,1])&&(startt<=dd2[hhh,2])&&(newcord1[hhh,9]==0))
                    {
                    fill2[startt:dd2[hhh,2]]=penalty
                    }
#5

                if ((lastt>=dd2[hhh,1])&&(lastt<dd2[hhh,2])&&(newcord1[hhh,9]==0))
                    {

                    fill2[dd2[hhh,1]:lastt]=penalty
                    }

                for (j in 1:length(dd2[,4])){############## Bonus ###########3
                    if ((startt<=dd2[hhh,1])&&(lastt>=dd2[hhh,4])&&(hhh!=j)
                            &&(startt<=dd2[j,1])&&(lastt>=dd2[j,2])&&(newcord1[hhh,9]==0))
                    {
                    z=length(dd2[j,4]:dd2[hhh,1])
                    fill2[dd2[hhh,1]]=fill2[dd2[hhh,1]]+cc*(1-z/100)
                    }
                }
            }

#----------bonus
```

```
        sumscore[lp]=sum(fill2)
        startt=startt+1
        lp=lp+1
        }

summax1=cbind(sumscore,sumcoord)
comodi=sumcoord[sumscore==max(summax1[,1])]
comodi2=sumlastt[sumscore==max(summax1[,1])]

#######################################

if (start0<=dd2[gh,1]){
Lower=comodi[1]
Upper=comodi2[1]
}
Upper=ifelse((Upper>=length(E)),length(E),Upper)
newcord1[gh,5]=Lower
newcord1[gh,6]=Upper

#-overlap ----------------------------------------**********************************
if(length(na.omit(newcord1[,5]))>1){

        for (iss in 1:(length(na.omit(newcord1[,5]))-1)){
                for (js in (iss+1):(length(na.omit(newcord1[,1])))){
                        if ((newcord1[iss,5]<=newcord1[js,1]) & (newcord1[iss,6]>=newcord1[js,2])&(newc
                                newcord1[js,7]=1
                                newcord1[iss,8]=1
                        }
                }
        }
for( i in 1:(length(na.omit(newcord1[,7])))){
        if((newcord1[i,7]==1)&(newcord1[i,8]==0)&(newcord1[i,9]==0)){
                newcord1[i,9]=1
        }
}
newcord1=orderBy(~-peakvalues, data=newcord1)
}
}
}

dimen=sum(newcord1[,9]==0)
newcord1=subset(newcord1,newcord1[,9]==0)
```

```
#----------------
#rare case.....remove duplicates further

if(length(na.omit(newcord1[,5]))>1){

        for (iss in 1:(length(na.omit(newcord1[,5]))-1)){
                for (js in (iss+1):(length(na.omit(newcord1[,1])))){
                        if ((newcord1[iss,5]<=newcord1[js,1]) & (newcord1[iss,6]>=newcord1[js,2])&(newc
                                newcord1[js,7]=1
                                newcord1[iss,8]=1
                        }
                }
        }
#---------------------------
newcord1=subset(newcord1, (newcord1[,7] == 0 & newcord1[,8] == 0)|
        (newcord1[,7] == 0 & newcord1[,8] == 1)|(newcord1[,7] == 1
                & newcord1[,9] == 1))
for( i in 1:(length(na.omit(newcord1[,7])))){
if((newcord1[i,7]==1)&(newcord1[i,8]==0)&(newcord1[i,9]==0)){
newcord1[i,9]=1
}
}
newcord1=orderBy(~-peakvalues, data=newcord1)
}
newcord1=subset(newcord1,newcord1[,9]==0)
#----------------



#-------------remove peakbottom
newcord=newcord1[,-c(4,7,8,9)]

if (dimen>1){
        newcord[,4]=ifelse(newcord[,5]==length(E),newcord[,5]-99,newcord[,4])
        newcord=orderBy(~Lower, data=newcord)

if (newcord[length(newcord[,1]),4]<newcord[(length(newcord[,1])-1),5]){
        newcord[length(newcord[,1]),4]=(newcord[(length(newcord[,1])-1),5])+1
}

newcord=orderBy(~-peakvalues, data=newcord)

for (pp1 in 1:(length(newcord[,1]))){
```

```
        for (pp in 2:(length(newcord[,1]))){
                if (newcord[pp1,5]==newcord[pp,4]){
                        newcord[pp,4]=newcord[pp,4]+1
                }
        }
}

for (pp in 2:(length(newcord[,1]))){
        if (newcord[pp,5]>n){
                newcord[pp,5]=n
        }
}

cut2=orderBy(~Lower, data=newcord)

if (cut2[(length(cut2[,1])),4]==cut2[(length(cut2[,1])-1),5]){
        cut2[(length(cut2[,1])),4]=cut2[(length(cut2[,1])),4]+1
}

numseg=length(cut2[,1])
}

if (dimen==1){
        newcord[4]=ifelse(newcord[5]==length(E),newcord[5]-99,newcord[4])

if (newcord[5]>n){
        newcord[5]=n
}
numseg=dimen
cut2=newcord
}

#------flag remove duplicate---

Lower=flagLow
Higher=flagHigh
longer_than100=rbind(Lower,Higher)

#----------------- Non overlapping
if ((dimen0>1)&&(dimen10>1)&&(dimen11>1)){

        cutN=0
```

```
        cutN=data.frame(cut[,4],cut[,5])

for(i in 1:(length(cut[,1])-1)){
        if (cut[i+1,4]<cut[i,5]){
                cutN[i+1,1]=cut[i,5]+1
        }
}

        cutN[length(cut[,1]),2]=min((cutN[length(cut[,1]),1]+99),n)
        cut[,4]=cutN[,1]
        cut[,5]=cutN[,2]
        cut[,4]=cutN[,1]
        cut[,5]=cutN[,2]
        cut=cut[,-c(6,7)]
}

if (dimen>1){
        cutPL=cut2[,4]
        cutPU=cut2[,5]
        notP2=0

if (cutPL[1]>0){
        notP2=cutPL[1]-1
}

if (cutPL[1]<0){
notP2=0}
coverP2=0
tti=1
tt=1

while (coverP2<100 & tt<=length(cutPL)-1)
{
        if((cutPU[tt]<cutPL[tt+1]) & tt<=(length(cut2[,1])-1))
                {
                notP2=notP2+cutPL[tt+1]-cutPU[tt]-1
                coverP2=100-(notP2/length(E))*100
        }
else
{
        coverP2=100-(notP2/length(E))*100
}
```

```
tt=tt+1
}

if (cutPU[length(cutPU)]<length(E))
        {
        notP2=notP2+(length(E)-cutPU[length(cutPU)]-1)
        coverP2=100-(notP2/length(E))*100
        }
}

if (dimen==1){
coverP2=(((cut2[5]-cut2[4])+1)/length(E))*100
}

#--------Printing Outputs----------------------------

old_method_coverage=coverP1
new_method_coverage=coverP2
number_of_segments_old=dimen0
number_of_segments_new=dimen
print("Length of the sequence")
print(n)
print("Centered Method")
print(cut)
print("Optimized Method")
print(cut2)
print("Centered Method_coverage")
print(old_method_coverage)
print("number_of_segments_Centered Method")
print(number_of_segments_old)
print("Optimized Method_coverage")
print(new_method_coverage)
print("number_of_segments_Optimized Method")
print(number_of_segments_new)
longer_than100

}#2

if (nrow(position)==2) {
        if (length(position[2,1]:position[2,2])>100){
                print(paste("There's only one inversion in the file",
                        "from",position[2,1], "to",position[2,2],"and greater than 100 bases"))
```

```
        }
}

#---Writing segments into files-------------------------------------------------

seqName2=substr(seqName,1,(nchar(seqName)-4))
path1=paste("C:/Users/Sameera/Documents/My Dropbox/Excursion
                            /Viruses/",seqName2,".txt",sep="")
SeqLetters=as.matrix(read.table(path1, header = FALSE,na=TRUE))
segment=0

#----------when there's only one inversion
if (nrow(position)==2){
        len=length(position[2,1]:position[2,2])
        add=(100-len)/2
        Lowe=position[2,1]-45
        Upp=position[2,2]+45

if (Lowe<1){
        Lowe=1
        Upp=100
}

segment=substr(SeqLetters,Lowe,Upp)
A=NA
A=cbind(Lowe,Upp,segment)
segName=paste(seqName,"_seg_Method1.txt",sep="")
path2=paste("C:/Users/Sameera/Documents/My Dropbox/Pen Drive
                /R Code/segments/",segName,sep="")

write.table(A,path2,append = FALSE, quote = FALSE, sep = " ",
            eol = "\n", na = "NA", dec = ".",
            col.names = FALSE, qmethod = c("escape", "double"))

print(paste("There's only one inversion in the file",seqName,"from"
                ,position[2,1], "to",position[2,2]))
fileName[lll]=segName
}
#-------------------------------------------------

if (nrow(position)>2){
        if ((dimen0>1)&&(dimen10>1)&&(dimen11>1)){
```

```
                for (pp in 1:length(cut[,1])){
                        segment[pp]=substr(SeqLetters,cut[pp,4],cut[pp,5])
                }

#------Saving Centered method--------------
A=NA
A=cbind(cut[,4],cut[,5],segment)
segName=paste(seqName,"_seg_Centered_Method.txt",sep="")
path2=paste("C:/Users/Sameera/Documents/My Dropbox/Pen Drive
                /R Code/segments/",segName,sep="")
fileName[lll+1]=segName

write.table(A,path2,append = FALSE, quote = FALSE, sep = " ",
            eol = "\n", na = "NA", dec = ".",
            col.names = FALSE, qmethod = c("escape", "double"))
}

if (dimen0==1){
        segment[]=substr(SeqLetters,cut[4],cut[5])

#-----------------------Save Centered method------------

A=NA
A=cbind(cut[4],cut[5],segment)
segName=paste(seqName,"_seg_Centered_Method.txt",sep="")
path2=paste("C:/Users/Sameera/Documents/My Dropbox/Pen Drive
                /R Code/segments/",segName,sep="")
fileName[lll+1]=segName

write.table(A,path2,append = FALSE, quote = FALSE, sep = " ",
            eol = "\n", na = "NA", dec = ".",
            col.names = FALSE, qmethod = c("escape", "double"))
}

#---------------------------------------

if (dimen>1){
#-----------------Save optimized method------------
segment2=NA

for (pp in 1:length(cut2[,1])){
        segment2[pp]=substr(SeqLetters,cut2[pp,4],cut2[pp,5])
```

```
}

B=cbind(cut2[,4],cut2[,5],segment2)
segName=paste(seqName,"_seg_Optimized_Method.txt",sep="")
path2=paste("C:/Users/Sameera/Documents/My Dropbox/Pen Drive
        /R Code/segments/",segName,sep="")
fileName[lll+2]=segName

write.table(B,path2,append = FALSE, quote = FALSE, sep = " ",
            eol = "\n", na = "NA", dec = ".",
            col.names = FALSE, qmethod = c("escape", "double"))
}

if (dimen==1){

#----------Saving optimized method-----------------
segment2=NA
segment2=substr(SeqLetters,cut2[4],cut2[5])

B=cbind(cut2[4],cut2[5],segment2)
B=data.frame(B,row.names=NULL)
segName=paste(seqName,"_seg_Optimized_Method.txt",sep="")
path2=paste("C:/Users/Sameera/Documents/My Dropbox/Pen Drive
                /R Code/segments/",segName,sep="")
fileName[lll+2]=segName

write.table(B,path2,append = FALSE, quote = FALSE, sep = " ",
            eol = "\n", na = "NA", dec = ".",
            col.names = FALSE, qmethod = c("escape", "double"))
}

#----------Saving regular method-----------------
segment3=NA
for (pp in 1:length(cut0[,1])){
segment3[pp]=substr(SeqLetters,cut0[pp,1],cut0[pp,2])
}
D1=cbind(cut0[,1],cut0[,2],segment3)
segName=paste(seqName,"_seg_Regular_Method.txt",sep="")
path3=paste("C:/Users/Sameera/Documents/My Dropbox
        /Pen Drive/R Code/segments/",segName,sep="")
fileName[lll+3]=segName
write.table(D1,path3,append = FALSE, quote = FALSE, sep = " ",
```

```
            eol = "\n", na = "NA", dec = ".",
            col.names = FALSE, qmethod = c("escape", "double"))

}
lll=lll+4
}
}#1

if (lenOut==1){
        ddBottom=ddBottom[-c(5)]
        ddcheck=ddBottom
        dd=ddBottom[-c(4)]
        cutPL=ddBottom[1]
        cutPU=min(ddBottom[1]+99,n)
print ("Only one segment")
print (c(cutPL,cutPU))
}

        }#loop
}#loop


fileName=na.omit(fileName)
fileName=fileName[!duplicated(fileName)]
path4=paste("C:/Users/Sameera/Documents/My Dropbox/Pen Drive
/R Code/segments/fileName_",segName3,".txt",sep="")
write.table(fileName,path4,append = FALSE, quote = FALSE, sep = " ",
            eol = "\n", na = "NA", dec = ".",
            col.names = FALSE, qmethod = c("escape", "double"))
```

Appendix D

R Script for Generating A Random RNA Sequence

```
#User defined function to generate a randome sequence
ran_seq=function(Pa,Pc,Pg,Pu,n){
#String of letters that should be randomly generated
a=c("A","C","G","U")
#Specifying the probabilities for the sampling
p=c(Pa,Pc,Pg,Pu)
#Sampling the string "a"
sample(a, n, replace = TRUE, prob =p)
}

#repeat the function for 100 times
for (i in 1:100){
#Generate
seq=as.matrix(ran_seq(0.25,0.25,0.25,0.25,1000))
seq=t(seq)
#saving the each sequence in a seperate file
path=paste("C:/Users/Sameera/Documents/My Dropbox/Pen Drive/Simulation/random",i,".txt",sep="")
write.table(seq,path,row.names=FALSE,col.names=FALSE,quote=FALSE,eol = "\n",sep="")
}
```

Curriculum Vitae

Sameera Dhananjaya Viswakula was born on August 04, 1983 in Colombo, Sri Lanka. He obtained his primary education from Gothami Junior School, Gampaha and he graduated from Nalanda College, Colombo in the Spring of 2002. He entered to the University of Colombo, Sri Lanka in Fall 2004 and obtained his Bachelor's degree in Statistics with first class honors in the Summer of 2008. Due to his significant work on undergraduate research, he was awarded the gold medal for the best undergraduate research in Statistics in 2008. Soon after the graduation, he joined to the Department of Statistics of University of Colombo as an instructor on record and worked there for a period of one year.

Sameera moved to El Paso in the Fall of 2009 for his Master's degree in Statistics at UTEP. While studying, he worked as a Teaching Assistant to the Department of Mathematical Sciences. He worked at the Center for Institutional Evaluation, Research and Planning of UTEP (CIERP) during the Summer of 2010 as a research assistant. Sameera is planning to start his doctoral studies in Statistics at Old Dominion University at Norfork, VA in Fall 2011.

Permanent address:

78A, New Road,

Ihalayagoda, Gampaha, Sri Lanka.