Secondary Structure Prediction Of Long RNA Sequences Based On Inversion Excursions And A Modularized Mapreduce Framework

Daniel Tesfai Yehdego
University of Texas at El Paso, dtyehdego@miners.utep.edu

Follow this and additional works at: https://digitalcommons.utep.edu/open_etd
Part of the Bioinformatics Commons, Biostatistics Commons, and the Computer Sciences Commons

Recommended Citation
https://digitalcommons.utep.edu/open_etd/1962

This is brought to you for free and open access by DigitalCommons@UTEP. It has been accepted for inclusion in Open Access Theses & Dissertations by an authorized administrator of DigitalCommons@UTEP. For more information, please contact lweber@utep.edu.
SECONDARY STRUCTURE PREDICTION OF LONG RNA SEQUENCES BASED ON INVERSION EXCURSIONS AND A MODULARIZED MAPREDUCE FRAMEWORK

DANIEL TESFAI YEHDEGO

College of Science
Computational Science Program

APPROVED:

__________________________
Ming-Ying Leung, Chair, Ph.D.

__________________________
Kyle L. Johnson, Ph.D.

__________________________
Olac Fuentes, Ph.D.

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

This work is dedicated to my beloved family, my Mother Negeset Woldu, my Father Tesfai Yehdego, and all Siblings.
SECONDARY STRUCTURE PREDICTION OF LONG RNA SEQUENCES BASED ON INVERSION EXCURSIONS AND A MODULARIZED MAPREDUCE FRAMEWORK

By

 DANIEL TESFAI YEHDEGO

THESIS
Presented to the Faculty of the Graduate School of
The University of Texas at El Paso
in Partial Fulfilment
of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Mathematical Sciences
THE UNIVERSITY OF TEXAS AT EL PASO
December 2012
Acknowledgements

I am grateful to my graduate and thesis adviser Dr. Ming-Ying Leung for her continues guidance and support throughout my study period. Her expertise, understanding, and patience are a considerable plus for my graduate studies. I would also like to thank Dr. Leticia Velazquez, Director of the Computational Science Program at the University of Texas at El Paso, for her support and encouragement to continue with my studies.

My gratitude also goes to Dr. Michela Taufer for hosting me at the GCL lab at the University of Delaware as a summer visiting student. It was an excellent experience and helped me introduce with people from different educational area of study. I would also like to thank to Boyu Zhang (GCL Lab at the University of Delaware) for her valuable support throughout my research span. I acknowledge the members of thesis committee, Dr. Kyle L. Johnson, Dr. Olac Fuentes, and Dr. Michela Taufer for taking their time to serve on my thesis committee and provide their valuable comments and suggestions.

I am indeed very grateful to Vikram Kumar Reddy, Sameera Viswakula, and Rahul Vegesna for their continues support of this work. I am grateful to the University of Texas at El Paso, Computational Science Program staff, faculty and students for the opportunity they offered me to study and work with them.

Daniel T. Yehdego
December 2012
ABSTRACT

Ribonucleic acid (RNA) molecules and their secondary structures play important roles in many biological processes including gene expression and regulation. The genomes of many viruses are also RNA molecules. Since secondary structures are crucial for RNA functionality, computational predictions of the RNA secondary structures have been widely studied. However, the tremendous demands on computer memory and computing time for complex secondary structures limit the capability of existing thermodynamically based algorithms for structure predictions to handling only short RNA sequences with a few hundred bases. One approach to overcome this limitation is by first cutting long RNA sequences into shorter, non-overlapping and manageable chunks whose secondary structures are predicted individually, and then assembling the resulting predictions for the chunks to give the structure of the original sequence.

The cutting process is a crucial component of this approach. Noting that all secondary structure elements, including stem-loops and pseudoknots, always contain an inversion, which is a stretch of nucleotides followed closely by its inverted complementary sequence, cutting methods based on inversion distributions have been proposed previously by our group. In this thesis, I compare three sequence cutting methods, called the centered, optimized, and regular methods, in terms of their capabilities to retain the prediction accuracy of the PKnotsRG algorithm after applying the cutting methods.

From the RFAM database, two sets of RNA sequences with known secondary structures have been selected as test data for the cutting methods. The first set contains 50 sequences without pseudoknots, and the second set contains 12 sequences with pseudoknots. The ratio between the prediction accuracy obtained with and without chunking is calculated over a range of inversion parameters, namely the minimum stem length $l$ and the maximum gap size $G$. With $l$ ranging from 3 to 8, and $G$ from 0 to 8, the maximum accuracy retention (MAR) percentage is obtained for each test sequence and each cutting method. We also experiment with varying the maximum chunk length $c$ between 60 and 300 and observe its influence on the MAR.
To systematically analyze the impact of the various cutting methods, predictions algorithms, and inversion parameters, we have established a modularized parallel computing framework using Hadoop MapReduce that enables us to automatically and efficiently explore large parametric spaces of chunking-, prediction-, reconstruction-, and analysis methods. To study the framework performance, we use a dataset of longer sequences consisting of seven RNA genomes of the viruses from the family Nodaviridae with lengths around 1300 or 3200 bases. Their secondary structures are not known, and because of their lengths, the use of MapReduce is vital for the exhaustive exploration of their possible secondary structures.

For the majority of test sequences, our results show that at least one of the cutting methods produces an MAR value greater than one, implying that the prediction accuracy of the PKnotsRG algorithm is actually improved by using the chunks instead of the whole sequence. Furthermore, the inversion based centered and optimized methods outperform the regular method that cuts the sequence naively in fixed length chunks. This suggests that our approach to secondary structure prediction of long RNA sequences by cutting is viable but the cutting should be performed intelligently by considering sequence features such as inversions.

The MapReduce performance analyses have also demonstrated that our approach can be implemented to run efficiently in the Hadoop MapReduce framework. This opens up possibilities to continue my research on exploring better models for secondary structure elements, testing the cutting methods with other prediction algorithms, and finding optimal values for inversion and chunk length parameters for prediction of secondary structures in long RNA sequences.
# Table of Contents

Acknowledgements........................................................................................................ v

ABSTRACT........................................................................................................................ vi

CHAPTER ONE .................................................................................................................. 1

INTRODUCTION ................................................................................................................ 1

1.1 Biological basics ........................................................................................................... 1

1.2 Importance of RNA secondary structure prediction .................................................. 1

1.3 Survey of existing RNA secondary structure prediction methods ............................ 3

1.3.1 Prediction based on graph theory .......................................................................... 4

1.3.2 Prediction based on information theory .................................................................. 7

1.3.3 Prediction based on genetic algorithms ................................................................. 8

1.4 Research objective ...................................................................................................... 9

CHAPTER TWO ................................................................................................................ 13

SECONDARY STRUCTURE PREDICTIONS FOR LONG RNA SEQUENCES.............. 13

2.1 Background and motivation ....................................................................................... 13

2.2 Inversions in RNA sequences ................................................................................... 14

2.3 Methods for sequence segmentation ......................................................................... 14

2.3.1 Regular cutting method ....................................................................................... 17

2.3.2 Centered cutting method ...................................................................................... 18

2.3.3 Optimized cutting method ................................................................................... 21

2.4 Structure prediction and assembly ............................................................................. 23

2.5 Assessment of segmentation methods ....................................................................... 24

2.6 Discussions and analysis of results ........................................................................... 24

2.6.1 Data set ................................................................................................................ 24

2.6.2 Accuracy retention (AR) ..................................................................................... 25

2.6.3 MAR comparison among cutting methods ............................................................. 27

2.6.4 Variation of MAR with inversion parameters ....................................................... 30

2.6.5 Variation of MAR with chunk length .................................................................... 32

CHAPTER THREE .......................................................................................................... 34

MODULARIZED MAPREDUCE FRAMEWORK TO SUPPORT RNA SECONDARY
STRUCTURE PREDICTION AND ANALYSIS WORKFLOWS .................................. 34

3.1 Motivation for using MapReduce .............................................................................. 34

3.2 Basics of the MapReduce paradigm ......................................................................... 36

3.3 Workflow for parallel chunk structure predictions .................................................... 37
3.4 Adapting multiple searching paths to MapReduce ................................................. 39
3.5 Granularity of mappers ......................................................................................... 41
3.6 Performance Analysis ......................................................................................... 42

CHAPTER FOUR ........................................................................................................ 49

CONCLUSIONS AND FUTURE WORK .................................................................... 49

4.1 Conclusions from work done to date ................................................................. 49
4.2 Further Investigations ......................................................................................... 50
4.3 Final Remarks ..................................................................................................... 53

Appendix .................................................................................................................... 54

References ................................................................................................................ 72

Biography .................................................................................................................. 75
CHAPTER ONE

INTRODUCTION

1.1 Biological basics

Ribonucleic acid (RNA) is part of a group of molecules known as the nucleic acids, which are one of the four major macromolecules (along with lipids, carbohydrates and proteins) essential for all known forms of life. RNA molecules play an active role in cells by catalysing biological reactions, controlling gene expression, communicating cellular signals, and synthesizing proteins.

Each RNA molecule is a single-stranded polymer made up of a sequence of nucleotides. A nucleotide consists of a nucleobase, a ribose sugar, and a phosphate group. There are four types of nucleobases used in RNA: adenosine (A), guanine (G), pyrimidine bases are cytosine (C), and uracil (U) [1]. The bases A and U tend to form hydrogen bonds with one another, so do C and G. This base-pairing has been discovered by James Watson and Francis Crick and therefore is called Watson-Crick base-pairing [2]. Beside these Watson-Crick base-pairs, the wobble base-pair, G : U is also possible in RNA molecules.

1.2 Importance of RNA secondary structure prediction

RNA structures can be categorized into three levels based on the arrangement of their nucleic acids. The sequence of linear bases in the nucleic acid length gives the primary structure of RNA. As we have seen earlier complementary bases can be paired according to the Watson-Crick pairing and this gives the secondary structure of a nucleic acid. The secondary structure, in addition, gives information about the complementary base pairing among the nucleotide sequence. The tertiary structure provides the detailed three dimensional (3D) coordinates for each atom of the RNA molecule.

The 3D structure of an RNA molecule is often the key to its function. Because of the instability of RNA molecules, experimental determination (e.g., by X-ray crystallography) of
their precise 3D structures is a time-consuming and rather costly process. However, useful information about the molecule can be gained from knowing its secondary structure, i.e., the collection of hydrogen-bonded base pairs in the molecule. Therefore, the secondary structure is normally predicted and verified by mutagenesis experiments before determining the tertiary structure. Moreover, it has been shown that RNA secondary structure prediction has applications to the design of nucleic acid probes [3]. It is also used by molecular biologists to help predict conserved structural elements in non-coding regions of gene transcripts [3]. Finally, there is also an application in predicting structures that are conserved during evolution [3].

**Figure 1 Two basic elements in RNA secondary structures**

Based on the base pair bonding, all RNA secondary elements can be categorized into two basic categories: stem-loops and pseudoknots (see Figure 1). Several base pairs follow each other to form a stem loop. RNA structures with Pseudoknots are formed when some parts of nucleic acid sequence covered by two base pairs are neither disjoint, nor have one contained in the other. It can also be seen from Figure 1 that any stem-loop or pseudoknot must contain an inversion, which is a string of nucleotides followed closely by its inverse complementary sequence. Figure 2 shows an example of inversion, with the 6-nucleotide string “ACCGCA” followed by its inverse complementary sequence “UGCGGU” after a gap of 3 nucleotides (gap size = 3).
Figure 2 Inversion with stem length 6 and gap size 3.

Although the presence of pseudoknots in the secondary structure was first recognized in the turnip yellow mosaic virus in 1982 [4], it is difficult to predict the presence of pseudoknots in RNA secondary structures. Because base pairing in pseudoknots is not well nested; that is, base pairs occur that "overlap" one another in sequence position, many popular computational tools for secondary structure prediction like Mfold, which were developed in the 1980’s, would not predict pseudoknot structures in a given RNA sequence.

The prediction of RNA secondary structure with pseudoknots by minimizing the energy is generally considered to be NP-complete [5, 6] or NP-hard, which implies it cannot be solved in polynomial time [7]. Hence it becomes impossible to simply find the optimal solution if the number of bases in the sequence is high. However, pseudoknot structure prediction in RNA molecules, with up to several hundred nucleotide bases seems to be an achievable goal [8], because the Watson-Crick pairs are a relatively simple and stereotyped interaction [8].

1.3 Survey of existing RNA secondary structure prediction methods

Many algorithmic theories have been proposed for the purpose of predicting RNA secondary structures. Most prediction algorithms are based on dynamic programming, graph theory, information theory or genetic algorithms. Some of the early methods developed are the Nussinov [9] and the Zuker algorithms [10]. Both make use of graph theory and dynamic programming to predict a pseudoknot-free folding of a sequence of bases. Generally, the number of possible secondary structure depends on the length of the input sequence in a
somewhat exponential relationship. Not all predicted secondary structures are biologically correct. Ideally, prediction algorithms should find all of the biologically correct secondary structures for a given RNA sequence.

1.3.1 Prediction based on graph theory

For a better analysis of the secondary structure of an RNA molecule, which can be viewed as a two dimensional (2D) network. Secondary structures can be represented using planar graphs and studied using graph theory, which is the branch of mathematics that deals with configurations described by nodes and connections. RNA secondary structures are modelled using the graph theory on the 2D plane consisting of vertices or nodes and edges or lines, which connect the vertices. In RNA, graphs are RNA secondary topologies where a vertex or an edge can represent multiple nucleotide bases or base pairs, which themselves are composed of multiple atoms and bonds. To allow graphical representation of complex RNA secondary topologies, special rules for defining RNA graphs are required. The rules specify how to represent RNA loops, bulges, junctions, and stems as vertices or edges in a graph [11]. Figures 3A and 3B show examples of tree (A) and non-tree (B) planar graphs. A tree is a connected graph whose vertex connections do not form closed paths. We exploit this and other tree enumeration theorems to count RNA's topological motifs. For RNA pseudoknots, non-tree graphs are required to describe their complex patterns of connectivity involving closed paths or faces (Figure 3B).

![Figure 3 tree (A) and non-tree (B) planar graphs for RNA structure representation](http://www.biomath.nyu.edu/oldrag/tutorials/graph_theory.html)

Both the Nussinov and Zuker algorithms use the fundamentals of the graph theory which describes the use of a dynamic programming algorithm together with backtracking to
predict the secondary structure of RNA molecules. The Nussinov algorithm uses a model which is based on a graph theory in such a way that the RNA secondary structure of long nucleotide sequence is represented by closing the nucleotide chain into a cyclic graph or planar loop [9]. In this planar loop representation, the vertices are identified with the bases in the chain. If the planar loop has a length of size \( n \), the vertices will be numbered consecutively from 1 to \( n \) and each numbered vertex is labelled by one of four bases (e.g., ‘A’, ‘C’, ‘G’, ‘U’) it represents. The edges of the planar loop representation will correspond to the base pairings in the folded form of the nucleotide chain. The folded chain should be planar so that the edges in the loop would not cross each other. Other assumptions we take when applying graph theory to the Nussinov algorithm is no base may be paired twice, so that each vertex belongs to at most one of the interior edges. In this case we are considering a special matching of the vertices on the loop. The Nussinov folding algorithm searches a secondary structure with the maximum number of base pairs [8]. It is a simple and efficient dynamic programming algorithm. The underlying assumption is that the more base pairs there are in a structure the more stable the structure is and the more likely it is. The algorithm takes advantage of the fact that the optimization problem can be solved by breaking it down into smaller subproblems and solving them. The recursive solution to the problem is to calculate the best structure for a subsequence \( S[i \ldots j] \) with \( 1 \leq i \leq n \) and \( i < j \leq n \) (\( n = \text{length of } S \)) from the best structure for even smaller subsequences, which has been calculated before. The result of the maximum number of pairs is stored in a two-dimensional matrix \( \beta \) at \( \beta(i,j) \). The overall time complexity of the recursion is \( O(n^3) \) and the complexity of space is \( O(n^2) \).

The Zuker algorithm (Zuker et al. 1981) is also a dynamic programming algorithm and it works on the basis of identifying the globally minimal energy structure for a sequence. The Zuker algorithm uses a graph theory for secondary structure representation, which are virtually identical to those used by the Nussinov algorithm [10]. Although the graph theory
representation are not essential to the Zuker algorithm, they help clarify precisely what structures are being considered and are also a good way to show how minimum energies of structures are computed. The nucleotides of RNA molecule are numbered from the 5’ end of the molecule. If $N$ is the total number of nucleotides, for $1 < i < N$, the $i^{th}$ nucleotide is denoted by $S_i$. Same as in Nussinov graphical representation of RNA secondary structure, the $N$ nucleotides of a sequence are referred to as vertices [10]. The $N-1$ arcs of the semicircle between the bases are called exterior edges. Line segments between nucleotides represent base pairing between two vertices. Zuker algorithm is more sophisticated than the Nussinov algorithm due to the fact that for every structural element an individual energy is calculated which then contributes to the overall energy of the structure. Therefore an energy function is defined for each of the loops elements, with the closing bond $(i, j)$:

1. energy of a stem is $eS(i, j, i + 1, j .. 1)$
2. energy of a hairpin loop is $eH(i, j)$
3. energy of an internal or a bulge loop is $eL(i, j, i', j')$
4. energy of a $k$ – multiloop is $eM(i, j, i_1, j_1, ..., i_k, j_k) = a + bk + ck'$, with $(i_1, j_1) ..., (i_k, j_k)$being the interior base pairs and $a, b, c$ = constants and $k'$ = number of unpaired bases in the loop.

A RNA secondary structure prediction program, PknotsRG [12], employs Zuker algorithms based on graph theory for finding the structure of minimal free energy. It computes the minimal free energy structure for a sequence, displayed as a dot bracket string. For better readability, base pairs involved in a pseudoknot are denoted with parenthesis for the first stem and square brackets for the second stem. When using PKnotsRG always remember due to the complexity of pseudoknot folding the run time of the program can be about a factor of the length of the sequence.
In graph-theory based RNA secondary structure prediction methods, common RNA secondary structure are automatically detected in a group of functionally or evolutionarily related RNA sequences. These methods find sets of stable stems conserved across multiple sequences and assemble compatible conserved stems to form consensus secondary structure motifs. This method doesn’t require the presence of global sequence similarity and it doesn’t also require prior structural alignment, and is able to detect pseudoknots structures. This method first find all possible stable stems in each sequence and compare them pairwise between sequences, and then find all potentially conserved stems shared by subsets of sequences, and finally assemble best sets of conserved stems to construct consensus secondary structure profiles, and report a number of them after structure refinement.

1.3.2 Prediction based on information theory

Some thermodynamic parameter values like for multiloops are not known yet and we cannot be sure that whether the thermodynamical parameters known so far are all totally correct. In that case an adapted version of information theory can be employed for secondary structure prediction [13]. The underlying assumption of this approach is that many RNA sequences of homologues are known and although they are different due to mutations during evolution, it is thought that they have a similar structure, since they function in a similar way. Therefore researchers assume that the mutations or differences in the sequence can be found in non-functional loop regions or that they are compensated by another mutation, so that the bases pair again. Shannon's information theory [14] with its entropy measure is used to make good predictions. Entropy is a measure of information and therefore every possible event has a certain probability and is assigned certain entropy according to the probabilities. If we think of RNA folding as a dynamic system of structures (states of the system), given much time, a sequence $S$ will form every possible structure $P$. For each structure there is a probability for
observing it at a given time, given its energy. This means we are looking for a probability
distribution of all possible structures. The lower the energy of the structure $P$, the more
probable the structure will be observed. The distribution of the structures (energies) in a
system can assume Boltzmann distribution. Let $X = \{X_1, \ldots, X_N\}$ denote a system of states,
where state $X_i$ has energy $E_i$. The system is Boltzmann distributed with temperature $T$ if only
if $P_i[X_i] = e^{-\beta E_i}/Z$ for $Z = \sum e^{-\beta E_i}$ where $\beta = (K_B T)^{-1}$. $\beta$ and $K_B$ are inverse temperature
and Boltzmann constant, respectively [14]. The reason we are using Boltzmann distribution is
it makes the least number of assumptions and it is the distribution with the lowest information
content/maximum (Shannon) entropy. The Boltzmann’s distribution can be easily adapted to
calculate the probability of a structure $P$ for $S$. $P_r[P/S] = e^{-\beta E_r}/Z$. More interesting, we can
even compute the probability of structural elements: $P_r[(i,j)/S] = \sum_{P_{3i,j}} P_r[P/S]$ . We
should make sure we calculate an efficient computation of partition function $Z$.

1.3.3 Prediction based on genetic algorithms

As the number of bases in a given sequence increases, the different thermodynamic
parameters also increase exponentially and this will be a problem when we use graph theory.
To overcome such problems, it’s advisable using heuristic methods such as genetic
algorithms (GAs), which uses biological techniques such as mutation, selection and crossover
or Monte Carlo methods, which are also used for making good predictions [14]. GAs were
initially introduced in the 1970s [15], and became popular in the late 1980s [16] for the
solution of various hard computational problems. This computational method, which is based
on evolutionary and biological principles, was reintroduced into the realm of biology and to
structural biology problems in particular, in the 1990s. GAs operate on a set of individuals
over many generations to get better solutions over time. Moreover it is guaranteed that its
solutions do get better over time and not worse [17]. Because it is not guaranteed these
heuristic methods find an optimal solution, Gerhard Steger suggests using them only when
deterministic or analytical methods have failed [13]. Predicting structures of RNA molecules
with pseudoknots are difficult and they usually take a considerable amount of time, but since
genetic algorithms come up with good results within a relatively small amount of time, they
are a good approach to find solutions for NP-complete problems.

The prediction of secondary structures of long RNA sequences is time and space
intensive, especially when the structures contain pseudoknots [18]. Memory and time
requirements for the prediction of RNA pseudoknots grow rapidly with the sequence length.
This issue has attracted a large body of bioinformatics work, where all approaches either
abandon the model of free energy minimization, or make restrictions on the class of
pseudoknots that can be recognized. The well-known algorithm by Rivas and Eddy [8],
which is able to predict a restricted class of pseudoknots, needs $O(n^6)$ time and $O(n^4)$
memory space, where $n$ is the sequence length. Even more restrictive, but more efficient by
two orders of magnitude is the program PKnotsRG by Reeder and Giegerich [24], requiring
$O(n^4)$ time and $O(n^2)$ space.

1.4 Research objective

Although computational predictions for simple stem-loop structures are well established,
the tremendous demand on computer memory and computing time resources for complex
secondary structures including e.g., pseudoknot predictions remains a challenge today for
long RNA sequences [19]. Most of the programs available for pseudoknot structure
prediction can only process sequences of limited lengths on the order of several hundred
bases at most. For example the approximate length limits we have encountered with four
prediction programs range from 200-800 nucleotide bases. These programs, therefore,
cannot be applied directly to larger RNA molecules such as the genomic RNAs of viruses
[19], which may be thousands of bases in length (e.g., RNA from the genomes of the Nodavirus range from 1306-3205 nucleotide bases. We approach the problem of predicting secondary structure of long RNA sequence by segmenting the nucleotide sequence into shorter non-overlapping chunks, predicting the secondary structures of each chunk individually, and then assembling the prediction results to give the structure of the original sequence.

High performance computing (HPC) can be used to distribute work across a cluster of machines which access a shared file system. For long RNA molecule structure prediction, we have proposed a segmentation approach by cutting shorter chunks and distributing the chunks to be processed by a cluster of parallel computers. For a systematic, fast prediction and analysis of the impact of the several segmentation methods and various parameter combinations, . With this context, my work in this thesis addresses two specific objectives:

1. **We will evaluate the capability of several segmentation methods in terms of retaining the secondary structure prediction accuracy by comparing the assembled structure with the structure predicted using the whole sequence without chunking.**

   Given a prescribed maximum chunk length $c$, one can cut the sequence regularly at every $c$ bases starting from one end. Although this regular cutting method is simple, it does not take into account any sequence feature that is essential for secondary structure formation. It may often cut into a secondary structure element, downgrading the quality of the final predicted structure.

   Our group has developed two other cutting methods, the centered and optimized methods, for segmenting long RNA sequences. Both methods identify regions in the sequence with high concentrations of inversions and avoid cutting into these regions. In the centered method, the longest spanning inversion clusters are centered in the chunks, while in
the optimized method, the number of bases covered by inversions is maximized. Both methods rely on first finding the location of all inversions. These locations may vary with the minimum stem length ($l$) and maximum gap size ($G$) between the two complementary arms of an inversion.

We evaluate the capabilities of the three cutting methods to retain the secondary structure prediction accuracy of the PknotsRG algorithm using the whole sequence for the prediction with a dataset of 50 RNA sequences with known structures available in the RFAM database. The ratio between the accuracies of the predicted structure with and without segmentation provides a measure of accuracy retention (AR) of the cutting method for the sequence.

2. Implement a parallel computing approach using the modularized framework MapReduce that enables us to automatically and efficiently explore large parametric spaces of chunking-, prediction-, reconstruction-, and analysis methods.

To understand the impact of various chunking, prediction, reconstruction, and analysis algorithms without worrying about implementation details, we propose a framework based on MapReduce. The workflow for a parallel chunk-based RNA secondary structure prediction consists of the following four steps: (1) chunking: each RNA sequence is cut into multiple chunks (or segments according to various chunking algorithms and parameters; (2) prediction: the secondary structure for each chunk is predicted independently by using one or multiple prediction programs; (3) reconstruction: the whole secondary structure of a sequence is reconstructed from predicted structures, one for each chunk; and (4) analysis: predicted secondary structures are compared versus e.g., experimentally observed structures or structures built by homology. Once we describe the workflow through an XML file, the MapReduce frame work executes the workflow in parallel automatically. In this thesis, we present our framework and we use it to study relevant accuracy and performance aspects
associated to chunk-based parallel predictions of secondary structures given a wide range of RNA sequences.

MapReduce is a programming model for processing large data sets [20]. It is a framework for processing parallel problems across huge datasets using multiple computers (nodes), collectively referred to as a cluster or a grid [20]. It requires implementations of the map and reduce. Distributed implementations of MapReduce require a means of connecting the processes performing the Map and Reduce phases. This may be a distributed file system, a direct streaming from mappers to reducers, or for the mapping processors to serve up their results to reducers that query them [21].

The rest of the thesis is organized as follows: chapter two introduces the secondary structure prediction of long RNA sequences based on inversion excursions. This chapter discusses inversions in RNA sequences, methods for sequence segmentation, structure prediction and assembly, assessment of segmentation methods and an overview of the Hadoop/MapReduce framework; chapter three presents and discusses a modularized MapReduce framework to support RNA secondary structure prediction and analysis workflows; and chapter four concludes the thesis and outlines future planned works.
CHAPTER TWO

SECONDARY STRUCTURE PREDICTIONS FOR LONG RNA SEQUENCES

2.1 Background and motivation

To overcome the tremendous demand on computing resources posted by pseudoknot prediction, various alternative algorithms have been proposed that restrict the types of pseudoknots to be predicted to keep computation time and storage size under control. Yet, most programs available to date for pseudoknot structure prediction can only process sequences of limited lengths, on the order of several hundred nucleotides. For example the approximate length limits we have encountered with four prediction programs range from 200-800 nucleotide bases. These programs, therefore, could not be applied directly to larger RNA molecules such as the genomic RNAs in viruses, which may be thousands of bases in length (e.g., RNA from the genomes of the Nodavirus range from 1306-3205 nucleotide bases.

In previous work of the group, we proposed to approach this problem by cutting a long RNA sequence to shorter non-overlapping chunks, predicting the secondary structures of each chunk individually by distributing them to different processors on a Condor grid, and then assembling the prediction results to give the structure of the original sequence [18]. We used this approach on the genome sequences of the virus family Nodaviridae, leading to the discovery of secondary structure essential for RNA replication of Nodamura virus [22]. The advantage of the grid computing approach is that it can accommodate a variety of existing and new prediction algorithms in a heterogeneous workflow. We immediately identified the challenge in the necessity of having a good segmentation strategy for cutting the sequence so that the predicted results of the chunks can be assembled to generate a reasonably accurate structure for the original molecule. In the past work we performed an exhaustive
search for all the possible ways to cut a sequence of bases. In the current study, we move away from this exhaustive approach and apply cutting methods based on statistical information on inversions. The approaches in this thesis outperform our previous work in terms of computing efficiency and confirm the capability of appropriate sequence segmentation methods to retain RNA secondary structure prediction accuracy.

2.2 Inversions in RNA sequences

Given a long RNA sequence, we identify regions with high concentration of inversions, as defined in section 1.2, by using an adapted version of the “Palindrome” program in the EMBOSS package [23]. EMBOSS is a free open source software analysis package. Two main reasons for adapting the EMBOSS palindrome program are: (i) the program works correctly on DNA but not RNA sequences, and (ii) we would like to, in our future work, allow for the G-U pairing, a feature not available in the EMBOSS Palindrome program. Our adapted program, InversFinder, is written in Java and is available for download at http://rnavlab.utep.edu. InversFinder requires a text file containing the RNA sequence in FASTA format as input. The minimum stem length $l$, and maximum gap size $G$ of the inversion are parameters specified by the user.

2.3 Methods for sequence segmentation

Our cutting methods rely on a general excursion approach first formulated in [24], which has already been applied to a variety of sequence analysis problems but not RNA secondary structure prediction. In many bioinformatics applications, the problem calls for identifying high concentration regions of a certain property in the nucleotide bases of biomolecular sequences. For example, replication origins in viral genomes have been predicted by looking for regions that are unusually rich in the nucleotides A and T in DNA
sequences [25]. In this thesis, we follow the same approach for RNA sequences, with the property of interest being whether or not the nucleotide base is found inside an inversion. We refer to the excursions generated by this property as “inversion excursions.”

The excursion method requires assigning to each nucleotide a positive score if it is a part of an inversion (including the two stems and the gap between them), and a negative score if it does not. We go through the entire nucleotide sequence accumulating the scores to form inversion excursions.

\[
\begin{align*}
\cdot & \cdot G C G A U U G C C G U C A G G C A A U A C U \cdot \\
\cdot & \cdot 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 \cdot 
\end{align*}
\]

**Figure 3** The binary sequence around an inversion

To facilitate the analysis, we use a parsing program to convert a RNA sequence into a binary sequence with the same length. If a nucleotide base is included in an inversion identified by the InversFinder program, it is given a value 1; if not, it is assigned a value of 0, as illustrated in Figure 3. Each “1” in the binary sequence is given a score of 1, and each “0” a negative score \(s\) which is determined as follows.

Consider the binary sequence as a realization of a sequence of independent and identically distributed (i.i.d.) random variables, \(X_1, X_2, \ldots, X_n\), where \(n\) is the length of the RNA sequence. These random variables take values either 1 or \(s\). Let \(p = Pr(X_i = 1)\) and \(q = 1 - p = Pr(X_i = S)\). The parameter \(p\) is naturally estimated by the percentage of bases contained in one or more inversions in the RNA sequence, i.e., the percentage of “1”s of the binary sequence. We require that the expected score per base \(u = p + qS\) to be negative. This requirement prevents favouring long segments to be high scoring segments. As done in [26] and other applications, the value of \(S\) can be conveniently selected by giving \(u\) a value of -0.5, and then determining the value of \(S\) according to Equation (1).

\[ S = [(\mu - p)/q] \quad (1) \]
The excursion score $E_i$ at position $i$ of the sequence is defined recursively as in Equations (2) and (3).

\[
E_0 = 0 \tag{2}
\]

\[
E_i = \max (E_{i-1} + X_i, 0) \quad \text{for } 1 \leq i \leq n. \tag{3}
\]

An excursion starts at a point $i$ where $E_i$ is zero, continues with a number of rising and falling stretches of positive values, and ends at $j > i$ where $j$ is the next position with $E_j = 0$. The score then stays at zero until it first becomes positive again for the start of the next excursion. 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 4 shows an example of an excursion plot. Rising stretches in the plot indicates the presence of inversions.

![Excursion Plot Example](image)

**Figure 4** Example of an excursion plot indicating peaks, peak bottoms, and peak lengths.

After the excursion plot is obtained, we identify the positions, called peaks, where the excursion scores are local maxima. Then, the bottom of each peak, which is the last position with zero excursion score right before the peak, is located. 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 require chunk lengths to be smaller than a prescribed maximum $c$, peak lengths greater than $c$ have to be flagged and analyzed separately. Figure 4 also shows examples of peaks, peak bottoms, and peak lengths. To be used with the centered and optimized cutting
methods, the peaks are sorted in decreasing order based on their excursion scores. A flow chart for the above steps is shown in Figure 5.

![Flow chart for the steps in sequence segmentation](image)

Figure 5 Flow chart for the steps in sequence segmentation

2.3.1 Regular cutting method

The regular cutting method simply cuts the nucleotide sequence regularly into chunks of a specified maximum chunk length $c$ until the sequence is exhausted. For example, with $c = 100$, the sequence RF00209_A from the RFAM database with 379 bases will be cut into four chunks made up of nucleotide positions 1-100, 101-200, 201-300, and 301-379. Figure 6 shows the four segments produced by the regular cutting method on the RF00209_A sequence. The vertical lines indicate the cutting positions.
Figure 6 Chunks obtained using the regular method

This is the most convenient method of segmentation, which can be used as a reference method. Obviously, rising stretches in an excursion plot, which indicate the presence of inversions and are likely to be part of secondary structures, can often be cut by this method. As a result, it is relatively easy to lose important structure information. Intuitively, one expects that both the centered and optimized methods, which take the inversion locations into account when placing the cuts, will perform better in retaining the secondary structure information in the sequence.

2.3.2 Centered cutting method

The centered method cuts the sequence by identifying inversions and building the chunks around them. Our objective is to segment the RNA sequence in such a way to avoid losing structure information as much as possible by centering the longest spanning inversion clusters in the chunks.

After peaks are identified, they are sorted in decreasing order of their excursion values. The peak with the highest excursion value is considered first. Later, the second highest peak is considered and so on. We stop the segmentation when the chunks cover all the 1’s in the binary sequence. Otherwise, we repeat this process for the next peak outside of
any of the already established chunks and with the next highest excursion value, 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) have been included in the chunks, whichever occurs first. Overlapping chunks are adjusted so that any nucleotide base is captured by only one chunk, with priority given to the peak with a higher excursion score.

For the selected peak, the position of the inversions or peak length positions are centered within the max chunk length of $c$ bases – where $c$ is defined by the user. We start at the bottom of this peak and follow the excursion until it returns to 0 the very next time and locate the position of the very last peak before the excursion returns to 0. We take the sequence segment between the peak bottom, and the position of the very last peak and place the sequence segment in the center of the chunk as illustrated in Figure 7. Suppose this centered segment contains $x$ nucleotide bases, if $(c - x)$ is even, the resulting chunk will have $(c - x)/2$ bases on each side of centered segment. If $(c - x)$ is odd, we will adjust the lengths on each side to the integers below and above $(c - x)/2$, allowing one side (chosen at random) to have one more nucleotide base than the other.

![Figure 7 Centered cutting method where $x = peak length$](image)

As an example, we applied the method to a RNA sequence, e.g., the 379-base RNA sequence RF00209_A in the RFAM database. As showed in Figure 8, the sequence is segmented into six chunks using the centered cutting method. These six segments cover the entire sequence. Labels 1, 2, 3, 4, 5 and 6 in Figure 8 represent the six segments with decreasing order of peak excursion scores. After the peak scores are sorted, the peak with the
highest excursion score is considered first. In this example, we take the maximum chunk length \( c = 100 \) (we can also use other maximum chunk lengths e.g., 60, 80, 120). The highest peak is found at position 297 with peak bottom at 257. As there are other inversions after the highest scoring peak, we follow the entire excursion to the end at position 356. Locating the last peak in this excursion at 343, we center the sequence segment from 257 to 343 to produce the chunk covering the 100 positions from 250 to 349. After this, the second highest scoring peak at position 54 is considered and the above procedure is repeated. This time, the peak bottom is at position 19 and last peak before the end of this excursion is at position 70. Centering the segment consisting of positions 19 – 70 in a chunk of 100 would require 24 positions on each side, extending the chunk beyond the beginning of the sequence. We therefore adjust the chunk to start at position 1 instead.

Note also that during the segmentation process, we might get a chunk that overlaps with previously established chunks. In those cases, we have to reconcile the situation by reducing one of the chunk lengths. For example, after establishing the first two chunks (labelled 1 and 2 in Figure 8), the next highest peak to be processed is at position 114, with peak bottom at position 89. Centering this peak will produce a chunk from positions 52 to 151, overlapping with the chunk 2. We resolve such conflicts by giving priority to the chunk with the higher number of bases within completely contained inversions. With this rule, we give priority to chunk 2, and reduce chunk 3 to positions 101 – 151. The process continues for the remaining chunks 4, 5, and 6.
2.3.3 Optimized cutting method

In the optimized method, cutting points are decided by choosing a segment containing the peak in an optimal position that yields the highest inversion scores for the segment, which is defined as the total number of nucleotide bases contained in the inversions that are entirely within the chunk. For example, consider a peak with peak length spanning the nucleotide bases between i and j and then all the chunks of size c covering this peak. That is, all segments with length c between positions j – (c – 1) and i + (c + 1) are considered (see Figure 10). The chunk with the maximum inversion score is then selected. Beginning with the highest peak, the above process is repeated until either all the peaks are utilized or all the inversion regions of the sequence are contained in established chunks, whichever occurs first. When chunks overlap, the cut points are adjusted in a similar way to that described for the centered method. The optimized method ensures that peak length positions are included within a chunk, but not necessarily in the center of the chunk. As an example, we applied the optimized method to the same RF00209_A RNA sequence file from the RFAM database, as shown in Figure 10. The optimized method produced only 5 chunks covering all except the first 18 positions of the sequence.
Figure 9 Segments considered by the optimized cutting method for a peak spanning positions $i$ through $j$.

It can be seen from Figure 10 that cuts into those sequence segments with rising excursions scores preceding the peaks are avoided by this method. Also, the chunks produced by the optimized method cover only 96.3% of the sequence, leaving out those parts of the sequence where no inversions are found. Therefore, wasting of computing resources is minimal in the optimized method.

Figure 10 Chunks obtained using the optimized method.
2.4 Structure prediction and assembly

After the RNA sequence is cut into chunks and their structures predicted independently using the PKnotsRG algorithm, the results are assembled by concatenating the predicted structures. At the same time, we use the same prediction algorithm to make predictions on the entire sequence without cutting. Both the whole and assembled predicted structures are compared to the known structure to obtain their respective prediction accuracies so that we can assess to what degree the cutting method can preserve the prediction accuracy of the algorithm when applied without any segmentation. Figure 11 shows the RF00209_A nucleotide sequence along with the bracket view of its known secondary structure. In the bracket view representation, bases that are hydrogen bonded with other bases are represented by a “(” or a “)”) and a matching pair of “(” and “)” indicates that the bases at those positions are paired to be part of a secondary structure. Unpaired nucleotide bases are represented by a “:” (colon).

```
> RF00209_A.bpseq
UACGAGGUAGUCAUCUCGUUAACACGAAUUGGACAACAAAUAAUAAUUGGUUCAGGCCUCCUCUCCA
GCCAGCCGGCAUCUGCCGCAUGCAGCCCAUAGGACACAUACCCAAATAGGACGGCAACUAGCAUGUGCAGC
UCCUCUGGGUGUCUAAGUCUCAGUACGGAACUGCAGUCAUGCUAGUGAGCAAGCGCCACGCCACUCCAGAGA
UGCUACUGGAACAGGCAUCGCCACAGCAACCUUAAACCUUGCGGUGCGCUAGGGUGAAGCGCAACGAC
UGGAUGGGAGUACGAGUCUAGGGCGCGAGGCCACUAAUAGGCAGUAAAAACUCUGAGUACCA
UGGCAACUGAGAUG
(((((:(:::))))))(((((((((((((::))))))))))))(((((((((((((::)))))))))))))))))))))(((((((((((((::))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))*)((::))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))}}))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))):
```

Figure 11 RF00209_A sequence and its secondary structure from RFAM database
2.5 Assessment of segmentation methods

In order to assess the capability of our cutting methods in preserving prediction accuracies, we define the accuracy retention (AR) value as the ratio between AC and AW, respectively, where AC is the accuracy of the predicted structure assembled from the chunks and AW is the accuracy of the predicted structure obtained from the whole sequence. These accuracies are given by the percentage agreement of the predicted structure with the known real structure calculated as:

\[
100 \times \frac{[\text{Number of unpaired bases in agreement}]}{\text{sequence length}} + 2 \times \frac{\text{Number of pairs in agreement}}{\text{sequence length}}
\]

The AR value of a cutting method on a sequence gives an indication of how much of the secondary structure prediction accuracy is retained after segmentation of the sequence. This will be used for assessing the performance of the different cutting methods described above. Various statistical tests are applied in the performance analysis for the different cutting methods. We used the statistical functions provided by the R package [27] for these purposes.

2.6 Discussions and analysis of results

2.6.1 Data set

In this thesis, we seek to understand whether cutting and rebuilding RNA secondary structures is as accurate as predicting the structures from the whole sequence and whether the obtained predictions are ultimately close to the experimentally observed structures. To study the capability of our cutting methods to capture secondary structures correctly, we used a dataset of 50 sequences in RFAM database [28], for which we know their secondary structures (either experimentally or per homology) and for which the prediction code used
(i.e., PKnotsRG) can also predict the secondary structures as a whole. To study the capability of our cutting methods to capture secondary structures with Pseudoknots correctly, we used a dataset of 12 sequences in RFAM database, for which we know their Pseudoknotted secondary structures (either experimentally or per homology). Note that contrary to the cases in this thesis, cutting may be the only possible prediction approach when sequences are very long (e.g., viral genome RNAs are usually over 1000 bases long) and the computing resources are limited.

2.6.2 Accuracy retention (AR)

To estimate whether the quality of our results is accidental or due to the proposed cutting approach based on inversion excursions, we performed a non-triviality test in which we collect the AR for the designed centered and optimized methods as well as for the regular cutting method that uses only chunk size to cut the sequence. For the regular cutting method, we used a fixed segment length of 100 bases for each RNA sequence. With the centered and optimized cutting methods, the chunks obtained depended on the value of inversion parameters $l$ and $G$ - each chunk could be no longer than 100 bases. We also allowed a range of $l$ values from 3 to 8, and of $G$ values from 0 to 8, resulting in a total of 54 $(l, G)$ pairs. Note that for some $(l, G)$ pairs no inversion may be found; for these cases the cutting methods did not apply and we assigned them an AR value of 0. For all three methods, we obtained the AR for each of the 50 RFAM sequences in our dataset. Note that RNA sequences in the dataset have lengths ranging from 127 to 568. Figure 12 shows the plots of mean AR values which are plotted against the $l$-$G$ parameters. Comparing these means with the regular cutting methods, neither the centered nor optimized methods showed any statistically significant difference from the regular cutting method.
Table 1 shows the computed maximum accuracy retention (MAR) for each sequence and for each cutting method when using a maximum chunk length of 100 and the different \((l, G)\) pair combination for the centered and optimized methods. An AR value larger than 1 means that the secondary structure predicted by the cutting method producing that value is more similar to the actual experimental or homology structure than the secondary structure predicted by using the whole sequence. In 32 out of 50 sequences, the secondary structures obtained by at least one cutting method (either regular, centered or optimized) have MAR larger than or equal to 1. We also observed that the numbers of chunks for the individual sequences range from 2 to 6.

From the results of the regular cutting method in Table 1, we observed that for 20 of the 50 sequences, the AR value is larger than or equal to 1; the AR values range from 0.253 to 2.816, with mean and standard deviation 0.936 and 0.418 respectively. Our results suggest that the segmentation process, even with this simplistic regular method, does not cause a big loss of secondary structure prediction accuracy in the PKnotsRG algorithm. In some cases, the prediction accuracy is actually increased by the segmentation. The range is quite wide, showing that while the average performance is good, one cannot guarantee a high AR value for an individual sequence. A scatter plot of AR values versus sequence lengths is shown in Figure 13. The correlation between AR and sequence length is found to be statistically not
significant \((r^2 = 0.01529, p\text{-value}= 0.3922)\). We therefore do not expect significant decline in AR values with increase in sequence length.

![Figure 13 Scatter plot of AR values against sequence length](image)

### 2.6.3 MAR comparison among cutting methods

With each of the centered and optimized methods, one can compute up to 54 AR values, one for each \((l, G)\) pair, and the maximum among these AR values is recorded as the MAR value in Table 1. For the centered cutting method, the MAR values range from 0.5323 to 2.8947, and are larger than or equal to 1 in 28 \((56\%)\) of the 50 sequences. For the optimized method, the MAR values range between 0.5323 and 3.181and are larger than or equal to 1 for 27 \((54\%)\) of the 50 sequences.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Length</th>
<th>Regular</th>
<th>Centered</th>
<th>Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF00002_B</td>
<td>150</td>
<td>1.0651</td>
<td>1.0651</td>
<td>1.1430</td>
</tr>
<tr>
<td>RF00003_B</td>
<td>161</td>
<td>1.0570</td>
<td>1.1571</td>
<td>1.1428</td>
</tr>
<tr>
<td>RF00004_A</td>
<td>145</td>
<td>1.1177</td>
<td>1.1177</td>
<td>1.1177</td>
</tr>
<tr>
<td>RF00007_B</td>
<td>129</td>
<td>1.1250</td>
<td>2.2083</td>
<td>1.8750</td>
</tr>
<tr>
<td>RF00009_A</td>
<td>320</td>
<td>0.9444</td>
<td>1.6111</td>
<td>1.6111</td>
</tr>
<tr>
<td>RF00011_A</td>
<td>374</td>
<td>0.7549</td>
<td>1.1900</td>
<td>1.2850</td>
</tr>
<tr>
<td>RF00012_B</td>
<td>219</td>
<td>0.8831</td>
<td>0.8831</td>
<td>0.8831</td>
</tr>
<tr>
<td>RF00013_A</td>
<td>183</td>
<td>0.2530</td>
<td>0.7048</td>
<td>0.6807</td>
</tr>
<tr>
<td>Item</td>
<td>Value 1</td>
<td>Value 2</td>
<td>Value 3</td>
<td>Value 4</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>RF00015_B</td>
<td>141</td>
<td>0.8108</td>
<td>1.0451</td>
<td>0.9280</td>
</tr>
<tr>
<td>RF00017_B</td>
<td>305</td>
<td>0.3194</td>
<td>0.5323</td>
<td>0.5323</td>
</tr>
<tr>
<td>RF00023_B</td>
<td>363</td>
<td>1.1510</td>
<td>1.1274</td>
<td>1.0897</td>
</tr>
<tr>
<td>RF00025_A</td>
<td>152</td>
<td>0.7664</td>
<td>0.9253</td>
<td>0.8879</td>
</tr>
<tr>
<td>RF00028_B</td>
<td>344</td>
<td>1.1303</td>
<td>1.5617</td>
<td>1.5617</td>
</tr>
<tr>
<td>RF00030_A</td>
<td>297</td>
<td>0.9932</td>
<td>1.0897</td>
<td>0.8879</td>
</tr>
<tr>
<td>RF00035_B</td>
<td>338</td>
<td>0.9280</td>
<td>1.0897</td>
<td>1.0897</td>
</tr>
<tr>
<td>RF00036_A</td>
<td>152</td>
<td>0.7664</td>
<td>0.9253</td>
<td>0.8879</td>
</tr>
<tr>
<td>RF00040_B</td>
<td>338</td>
<td>0.7521</td>
<td>0.8635</td>
<td>0.8388</td>
</tr>
<tr>
<td>RF00045_B</td>
<td>213</td>
<td>1.0000</td>
<td>1.0000</td>
<td>0.9674</td>
</tr>
<tr>
<td>RF00050_A</td>
<td>157</td>
<td>0.9195</td>
<td>1.0162</td>
<td>0.9678</td>
</tr>
<tr>
<td>RF00100_A</td>
<td>330</td>
<td>1.4300</td>
<td>1.6802</td>
<td>1.6000</td>
</tr>
<tr>
<td>RF00102_B</td>
<td>159</td>
<td>0.7791</td>
<td>0.9767</td>
<td>0.8954</td>
</tr>
<tr>
<td>RF00168_B</td>
<td>179</td>
<td>0.7172</td>
<td>0.8966</td>
<td>0.8966</td>
</tr>
<tr>
<td>RF00171_A</td>
<td>168</td>
<td>1.0245</td>
<td>1.0245</td>
<td>1.0245</td>
</tr>
<tr>
<td>RF00177_A</td>
<td>482</td>
<td>0.5994</td>
<td>0.7467</td>
<td>0.7112</td>
</tr>
<tr>
<td>RF00182_A</td>
<td>129</td>
<td>0.8204</td>
<td>0.9229</td>
<td>1.2434</td>
</tr>
<tr>
<td>RF00193_A</td>
<td>273</td>
<td>1.0150</td>
<td>0.9700</td>
<td>0.9850</td>
</tr>
<tr>
<td>RF00209_A</td>
<td>379</td>
<td>0.6193</td>
<td>0.8945</td>
<td>1.0322</td>
</tr>
<tr>
<td>RF00210_A</td>
<td>462</td>
<td>0.8431</td>
<td>0.8708</td>
<td>0.9231</td>
</tr>
<tr>
<td>RF00225_B</td>
<td>151</td>
<td>1.0000</td>
<td>1.5229</td>
<td>1.5229</td>
</tr>
<tr>
<td>RF00230_B</td>
<td>246</td>
<td>1.0835</td>
<td>1.5003</td>
<td>1.6251</td>
</tr>
<tr>
<td>RF00231_A</td>
<td>275</td>
<td>1.4127</td>
<td>1.3254</td>
<td>1.3411</td>
</tr>
<tr>
<td>RF00232_A</td>
<td>170</td>
<td>1.2064</td>
<td>1.2609</td>
<td>1.2609</td>
</tr>
<tr>
<td>RF00234_B</td>
<td>160</td>
<td>0.9596</td>
<td>1.0302</td>
<td>1.1312</td>
</tr>
<tr>
<td>RF00252_B</td>
<td>182</td>
<td>0.8503</td>
<td>0.9048</td>
<td>0.9591</td>
</tr>
<tr>
<td>RF00264_B</td>
<td>134</td>
<td>0.5862</td>
<td>0.8736</td>
<td>0.8736</td>
</tr>
<tr>
<td>RF00290_A</td>
<td>140</td>
<td>0.6500</td>
<td>0.7571</td>
<td>0.7357</td>
</tr>
<tr>
<td>RF00373_A</td>
<td>133</td>
<td>1.1014</td>
<td>1.2899</td>
<td>1.2899</td>
</tr>
<tr>
<td>RF00387_A</td>
<td>168</td>
<td>0.3292</td>
<td>0.7683</td>
<td>0.6830</td>
</tr>
<tr>
<td>RF00389_B</td>
<td>158</td>
<td>0.844</td>
<td>1.1100</td>
<td>1.0367</td>
</tr>
<tr>
<td>RF00391_A</td>
<td>171</td>
<td>0.4561</td>
<td>0.6257</td>
<td>0.5848</td>
</tr>
<tr>
<td>RF00433_B</td>
<td>152</td>
<td>0.4094</td>
<td>0.8121</td>
<td>0.8121</td>
</tr>
<tr>
<td>RF00437_B</td>
<td>131</td>
<td>0.5699</td>
<td>0.828</td>
<td>0.7205</td>
</tr>
<tr>
<td>RF00444_B</td>
<td>144</td>
<td>1.1401</td>
<td>1.0561</td>
<td>0.9626</td>
</tr>
<tr>
<td>RF00451_B</td>
<td>309</td>
<td>1.0376</td>
<td>1.0753</td>
<td>1.2481</td>
</tr>
<tr>
<td>RF00463_A</td>
<td>127</td>
<td>2.8158</td>
<td>2.8947</td>
<td>3.1581</td>
</tr>
<tr>
<td>RF00484_A</td>
<td>149</td>
<td>1.9349</td>
<td>1.9349</td>
<td>1.8043</td>
</tr>
<tr>
<td>RF00487_B</td>
<td>211</td>
<td>1.2647</td>
<td>2.7204</td>
<td>2.5439</td>
</tr>
<tr>
<td>RF00488_B</td>
<td>568</td>
<td>0.877</td>
<td>1.2078</td>
<td>1.2694</td>
</tr>
<tr>
<td>RF00492_B</td>
<td>144</td>
<td>0.4554</td>
<td>0.8118</td>
<td>0.9109</td>
</tr>
<tr>
<td>RF00503_A</td>
<td>293</td>
<td>0.9946</td>
<td>1.2796</td>
<td>1.1452</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>-------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>RF00506_B</td>
<td>144</td>
<td>0.9861</td>
<td>0.9861</td>
<td><strong>1.0000</strong></td>
</tr>
<tr>
<td><strong>Mean</strong></td>
<td>226.98</td>
<td>0.9260</td>
<td>1.1492</td>
<td>1.1422</td>
</tr>
<tr>
<td><strong>STD</strong></td>
<td>106.38</td>
<td>0.4158</td>
<td>0.4726</td>
<td>0.4726</td>
</tr>
</tbody>
</table>

Table 1 MAR for individual data sequences

From Table 1, we also note that each cutting method performs better than the other methods in some but not all of the sequences. So, for each cutting method, we considered the data sequences where it gives the highest MAR. For those sequences in which there were ties (i.e., more than one method attains the highest MAR), the count was split equally among the methods. These counts are shown in Table 2, showing that the centered method produces the highest MAR in 51% of our data sequences, followed by the optimized (37%) and regular (12%) methods. This suggests that the centered and optimized methods attain higher MAR than the regular method and that the centered slightly outperforms the optimized method. We proceed to check whether there are indeed significant differences among the cutting methods using Friedman test (Chapter 5, Section 8 in [30]), a non-parametric ANOVA test for repeated measures. This test requires ranking the MAR among the three methods for each data sequence and obtaining the rank sum for each method over the entire dataset. Methods sharing the same MAR are assigned an equal averaged rank. For example, for RF00002_B (first line in Table 1), both the regular and centered methods are ranked 1.5, and the optimal method is ranked 3. The rank sums are shown in Table 3. The Friedman test results confirm our initial observation that the centered and optimal methods perform significantly better than the regular method ($p$-value < 0.0001), but there is no significant difference between the two methods themselves.
2.6.4 Variation of MAR with inversion parameters

While the $l$ and $G$ parameters have no bearing on the regular cutting method, the accuracy of both the centered and optimized methods depend on the choice of these parameters. The ($l$, $G$) parameter pair at which MAR is attained varies from sequence to sequence. Figure 14 shows the counts of sequences attaining MAR at different parameter values. If a sequence attains the same MAR value with multiple ($l$, $G$) parameter pairs, it will be counted for all these pairs. Some ($l$, $G$) pairs seem to be more likely to attain MAR than others. As the stem-length increases, it is observed that the possibility of attaining the MAR decreases. In particular, no sequence in the dataset attains MAR with $l > 6$. Also, the number of ($l$, $G$) pairs attaining MAR is higher for the centered method.

<table>
<thead>
<tr>
<th></th>
<th>Regular</th>
<th>Centered</th>
<th>Optimized</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>25.5</td>
<td>18.5</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 2 count of sequences with highest MAR for chunk length of 100.

<table>
<thead>
<tr>
<th></th>
<th>Regular</th>
<th>Centered</th>
<th>Optimized</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>66</td>
<td>120.5</td>
<td>113.5</td>
<td>300</td>
</tr>
</tbody>
</table>

Table 3 MAR rank sums.

![Number of sequences attaining highest MAR at various ($l$, $G$) parameter values.](image)
We also allowed a range of $l$ values from 3 to 8, and of $G$ values from 0 to 8, resulting in a total of 54 ($l$, $G$) pairs. For all three methods, we obtained the AR for each of the 12 RFAM sequences in our dataset. Note that RNA sequences in the dataset have lengths ranging from 62 to 451. Table 4 shows the computed maximum accuracy retention (MAR) for each sequence and for each cutting method when using a maximum chunk length of 60 and the different ($l$, $G$) pair combination for the centered and optimized methods. In 9 out of 12 sequences, the Pseudoknotted secondary structures obtained by at least one cutting method (either regular, centered or optimized) have MAR larger than or equal to 1.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Length</th>
<th>Regular</th>
<th>Centered</th>
<th>Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF00010</td>
<td>312</td>
<td>0.4183</td>
<td>0.8929</td>
<td>0.9183</td>
</tr>
<tr>
<td>RF00024</td>
<td>451</td>
<td>0.4535</td>
<td>0.8624</td>
<td>0.9368</td>
</tr>
<tr>
<td>RF00061</td>
<td>323</td>
<td>0.9285</td>
<td>1.6022</td>
<td>1.4489</td>
</tr>
<tr>
<td>RF00094</td>
<td>89</td>
<td>0.2941</td>
<td>0.4314</td>
<td>0.3726</td>
</tr>
<tr>
<td>RF00140</td>
<td>112</td>
<td>0.5199</td>
<td>1.2601</td>
<td>0.9601</td>
</tr>
<tr>
<td>RF00165</td>
<td>62</td>
<td>0.7498</td>
<td>1</td>
<td>1.3001</td>
</tr>
<tr>
<td>RF00216</td>
<td>302</td>
<td>1.449</td>
<td>2.3912</td>
<td>2.6954</td>
</tr>
<tr>
<td>RF00233</td>
<td>84</td>
<td>0.4912</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RF00259</td>
<td>169</td>
<td>0.6034</td>
<td>0.7413</td>
<td>1.1378</td>
</tr>
<tr>
<td>RF00261</td>
<td>221</td>
<td>0.5555</td>
<td>3.3114</td>
<td>2.4003</td>
</tr>
<tr>
<td>RF00458</td>
<td>202</td>
<td>0.6746</td>
<td>1.1687</td>
<td>1.2531</td>
</tr>
<tr>
<td>RF00507</td>
<td>79</td>
<td>0.6486</td>
<td>1.2432</td>
<td>1.2432</td>
</tr>
<tr>
<td><strong>Mean</strong></td>
<td>200.5</td>
<td>0.64887</td>
<td>1.3254</td>
<td>1.30555</td>
</tr>
<tr>
<td><strong>STD</strong></td>
<td>124.202</td>
<td>0.30132</td>
<td>0.79464</td>
<td>0.64347</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Regular</th>
<th>Centered</th>
<th>Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 4 MAR for individual data sequences with pseudoknots

Table 5 count of pseudoknotted sequences with highest MAR for chunk length of 60
2.6.5 Variation of MAR with chunk length

Unlike the \((l, G)\) parameters which can be freely chosen by the user, the max chunk-length \(c\) in the cutting method is limited by the particular secondary structure prediction program need. Just to explore the possible effect of \(c\) on MAR, for the cutting method, we run our program with maximum chunk length ranging from 60 to 150 bases; by increasing 10 bases each time, we obtain a total of 540 combinations per sequence. Table 4 represents the count of sequences with highest MAR over all chunk length combinations for each sequence and figure 15 also shows the plot of max chunk-length versus MAR for the three cutting methods. The result shows that the average MAR values stays approximately constant as the max chunk-length increases from 60-130 and start increasing as the max chunk-length increases from 130-150. To see a clear relationship between MAR and max chunk-length, we will increase the max chunk-length from 150-300 at a 10 nucleotide base and see what happens to MAR.

<table>
<thead>
<tr>
<th>Chunk Length</th>
<th>Regular</th>
<th>Centered</th>
<th>Optimized</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>5</td>
<td>17</td>
<td>25</td>
<td>47</td>
</tr>
<tr>
<td>70</td>
<td>5</td>
<td>22</td>
<td>21</td>
<td>48</td>
</tr>
<tr>
<td>80</td>
<td>2</td>
<td>19</td>
<td>27</td>
<td>48</td>
</tr>
<tr>
<td>90</td>
<td>2</td>
<td>25.5</td>
<td>20.5</td>
<td>48</td>
</tr>
<tr>
<td>100</td>
<td>6</td>
<td>25.5</td>
<td>18.5</td>
<td>50</td>
</tr>
<tr>
<td>110</td>
<td>1</td>
<td>9.5</td>
<td>6.5</td>
<td>17</td>
</tr>
<tr>
<td>120</td>
<td>0</td>
<td>8.5</td>
<td>8.5</td>
<td>17</td>
</tr>
<tr>
<td>130</td>
<td>0</td>
<td>8.5</td>
<td>6.5</td>
<td>15</td>
</tr>
<tr>
<td>140</td>
<td>0.33</td>
<td>7.33</td>
<td>8.33</td>
<td>16</td>
</tr>
<tr>
<td>150</td>
<td>0</td>
<td>6.5</td>
<td>7.5</td>
<td>14</td>
</tr>
</tbody>
</table>

Table 5 Count of sequences with highest MAR over chunk length range of 60-150
This chapter has outlined the potential of secondary structure prediction using sequence segmentation and shows that cutting methods with considerations of inversion locations outperform those without. The chapter also raises further questions such as what are the best $(l, G)$ parameters and chunk-length $c$ that can produce high AR values for a given sequence. This problem is work in progress by our group.
CHAPTER THREE

MODULARIZED MAPREDUCE FRAMEWORK TO SUPPORT RNA SECONDARY STRUCTURE PREDICTION AND ANALYSIS WORKFLOWS

3.1 Motivation for using MapReduce

Due to new computational challenges (e.g., in next generation sequencing [30, 31]), high performance computing (HPC) has become increasingly important in bioinformatics data analysis. HPC typically involves distribution of work across a cluster of machines which access a shared file system, hosted on a storage area network. Work parallelization has been implemented via application programming interfaces (APIs) such as the Message Passing Interface (MPI) and, more recently, Hadoop’s MapReduce API which is considered a case of cloud computing. In brief, cloud computing in HPC is “a large pool of easily usable and accessible virtualized resources such as hardware, development platforms, and/or services that provides computation, software applications, data access, data management and storage resources without requiring users to know the location and other details of the computing infrastructure” [32, 33, 34]. The application of cloud computing in biomedical research is increasing since its inception. Many research lab institutes, as well as biotech, and pharmaceutical companies, are considering cloud computing as a cost-effective alternative to process and store vast amounts of data generated in the post-genomic era.

From the previous chapter, by cutting a long RNA sequence into chunks, predicting the chunks independently, and reconstructing the secondary structure from the smaller structures, we have found out that the accuracy of the predicted secondary structure from the segmentation method can yield better results comparing to the structure predicted using the entire RNA sequence as a whole.

The process of rebuilding complex secondary structures and analysing their accuracies can generate sophisticated workflows consisting of several steps, i.e., (1) cutting
the entire RNA sequence into multiple segments (chunks) based on different cutting strategies and chunk properties such as the maximum chunk-length $c$ or the number and length of inversions in the chunk; (2) prediction of secondary structures of the chunks using existing prediction codes; (3) reconstruction of the whole secondary structure from the chunk predictions; and (4) quantify the accuracy of the reconstructed prediction for statistical analysis. Each step can be seen as an optimization process in which several algorithms and parameter settings can be selected, each with different degrees of accuracy and performance.

The chunk-based prediction of RNA secondary structures for long RNA sequences can be performed in parallel, thus benefiting from parallel computing systems and paradigms. Writing traditional parallel programs using message passing paradigms such as MPI requires the programmer to handle data partition and distribution, inter process communications, load balancing, and fault tolerance problems explicitly. Alternative parallel paradigms such as MapReduce [35] allow the programmer to express more easily algorithm logic in two functions: map and reduce. A runtime system implementing the MapReduce paradigm such as Hadoop can take care of the run-time details for the researchers. To understand the impact of various chunking, prediction, reconstruction, and analysis algorithms without worrying about implementation details, we propose a framework based on MapReduce. The workflow can be described through an XML file, and the MapReduce framework executes the workflow in parallel automatically. Our framework enables a search of a much larger sample space in which different chunk-based methods are examined including both non-overlapping and overlapping methods. In this thesis we present our framework use it to study relevant accuracy and performance aspects associated to chunk-based parallel predictions of secondary structures given a wide range of RNA sequences.
3.2 Basics of the MapReduce paradigm

MapReduce is a parallel programming model that facilitates the processing of large distributed datasets. It was originally proposed by Google to index and annotate data on the Internet [36]. In this paradigm, the programmer specifies two functions: map and reduce. The map function takes as input a key and value pair \((k1, v1)\), performs the map function, and outputs a list of intermediate key and value pairs list \((k2, v2)\) which may be different from the input.

The runtime system automatically groups all the values associated with the same key \(k2\) and forms the input to the reduce function. The reduce function takes as input a key and values pair lists\((k2, list(v2)\)\), performs the reduce function, and outputs a list of values. Note that the input values \(v2\) are the list of all the values associated with the same key \(k2\).

\[
\text{(Reduce } (k2, list(v2)) \rightarrow list(v3))\text{.}
\]

While the map function is the function that will be applied to the input key and value pair \((k1, v1)\), there are also map tasks (aka mapper) which refer to the processes that run on the processors performing the map functions on \((k1, v1)\).

Hadoop is one way of implementing a MapReduce software framework. It can easily be installed on a commodity Linux cluster to permit large scale distributed data analysis. Hadoop is designed to run on a large number of machines that don’t share any memory or disks. The application programs’ execution across the nodes of the cluster can also be controlled using Hadoop with Job and Task Trackers. Hadoop tries to automatically allocate the data with the computing node. The Hadoop MapReduce implementation has a principal advantage of performance, that is, Hadoop schedules Map tasks close to the data on which they will work, with “close” meaning the same node or, at least, the same rack. Note that Hadoop is not the only MapReduce framework available to the public but its open-source,
easy-to-use, modular, and robust implementation makes it an appealing MR framework among the several frameworks available.”

MapReduce is appealing to scientific problems, including the one addressed in this thesis, because of the simplicity of programming, the automatic load balancing and failure recovery, and the ease of scaling nature. It has been widely adapted for many bioinformatics applications, e.g., Hong et al. [36] designed a RNA-Seq analysis tool for the estimation of gene expression levels and genomic variant calling; Langmead et al. [37] designed a next-generation sequencing tool based on Hadoop MapReduce. To the best of our knowledge, this work is the first one to adapt MapReduce into secondary structure predictions of long RNA sequences. This thesis shows how the MapReduce framework allows us to gain insights about different chunking strategies easily, accurately, and efficiently.

### 3.3 Workflow for parallel chunk structure predictions

Rather than predicting the RNA sequence as a whole, we cut it into chunks and predict each chunk independently before merging the predictions into the whole secondary structure. The chunking process can be performed in different ways and this can result in large searches for effective ways to chunk sequences as well as a large number of independent prediction jobs that can potentially be performed in parallel. The workflow for a parallel chunk-based RNA secondary structure prediction consists of the following four steps: (1) segmentation: each RNA sequence is cut into multiple chunks (or segments) according to various cutting methods and inversion parameters; (2) prediction: the secondary structure for each chunk is predicted independently by using one or multiple prediction programs; (3) reconstruction: the whole secondary structure of a sequence is reconstructed from predicted structures, for the chunks; and (4) analysis: predicted secondary structures are compared versus experimentally observed structures. Figure 15 shows the prediction workflow. Note
that the chunks do not necessarily have the same length: the lengths depend on the cutting method and parameters used. Also note that the chunk’s prediction time and memory usage can vary based on the nucleotide sequence of the chunk and the prediction program used. Finally, note that in most prediction codes the time and memory used do not grow linearly but exponentially [38] with the number of nucleotides, with the exponential factor depending on the code complexity and its capability to capture complex RNA secondary structure such as pseudoknots.

Figure 15 Workflow of the chunk-based RNA secondary structure prediction framework (a) and example of searching path (b).
3.4 Adapting multiple searching paths to MapReduce

When predicting RNA secondary structures, we often have to contain the maximum sequence lengths (i.e., the maximum number of nucleotides bases the sequence can have) to meet both memory and time constraints on the available computing node. As we are now working with chunks, we will use the maximum chunk length $c$ parameter as synonymous of node limitations (both computing and memory limitations) for a given prediction code and computing node. Note that given a sequence, its chunks can be equal to or shorter than based on the chunking method. In our tests, $c$ ranges between 60 to 300 bases representing the range of sequence lengths that most existing prediction codes can deal with on desktop machines. More values can be explored by setting the initial parameters and without changes to the framework. As before, we vary the minimum stem length $l$ and maximum gap size $G$ for the centered and optimized method in our search; their values range from 0 to 8 for $l$ and 3 to 8 for $G$

Given an RNA sequence, the search for the best set of chunking parameters (i.e., $c$, $l$, $G$, and chunking method) requires traversing or searching a multi-level tree (i.e., the chunking tree in Figure 15.b). The overall workflow naturally adapts to fit into the MapReduce (MR) paradigm and can be easily implemented with Hadoop [39] for which the chunking and predictions can be solved by multiple mappers while the reconstruction and the analysis is done by a single reducer.

More generally, each MR job is designed to partially traverse the multi-level tree. Multiple MR jobs can be executed in parallel to explore the whole tree. The multiple searching paths combine attributes of both breadth-first search (performed by multiple MapReduce jobs in parallel) and depth-first search (performed by a single MapReduce job). While traversing the tree with multiple MR jobs, we can explore the impact of different chunking methods as well as different $c$, $l$, $G$ parameters for a given sequence. An example of an MR job is shown in the circled part in Figure 15.b for which we assume the centered
chunking method, with \( c = 60 \) bases, and we vary the stem and gap lengths between 0 and 8 and between 3 and 8 respectively.

As previously outlined, for a sequence and a combination of parameters, the mappers perform the chunking and predictions. The input to each mapper is a \( \langle k1, v1 \rangle \) value pair, in which \( k1 \) is the id of the sequence, and \( v1 \) is the chunking parameters values (including the chunking method). The mapper cut the sequence according to the chunking parameters values in the chunking step by identifying a variable number of chunks meeting the parameter requirements (e.g., in terms of max or min lengths). Note that different combinations of parameters (each branch of the tree) can result in a different numbers of chunks. (See example in Figure 15.b).

The prediction is performed on one or multiple chunks of each tree leaf by using a prediction code. Here we use the secondary structure prediction code developed by Reeder and Giegerich (PKnotsRG) [40] but any other code could be easily used in our framework. After the prediction, each mapper output the list of \( \langle k1, v1 \rangle \) pairs as intermediate output to reduce. The \( k2 \) is the id of the whole secondary structure to which the predicted chunk belongs, and \( v2 \) is the predicted secondary structure of the chunk. After the Hadoop runtime system groups all the values associated with the same key and pass the \( \langle k2, list(v2) \rangle \) to the reducer, the reducer reconstructs the whole secondary structure of the sequence using all the \( v2 \) (predicted chunk structures) associated with the same \( k2 \). If required, the reducer analyzes the results in terms of their accuracy.

Currently, our framework simply glues all the predicted secondary structures and generates the secondary structure for the whole sequence. This is possible because the chunking does not allow any overlapping between two consecutive chunks. More sophisticated reconstruction methods that include partial chunk overlapping can be used with minor change to our framework. The framework also allows us to analyze the secondary
structures and compare them to each other, to the same prediction built from the whole sequence, and to secondary structures built experimentally or per homology.

3.5 Granularity of mappers

In general a mapper is the process that runs on a processor which applies the map function to a specific key and value pair. In our framework, each mapper runs the chunking program and then the prediction program on the RNA sequence with the parameter values. Each mapper explores one branch of the tree and generates the set of segments as the output of the chunking program, then it predicts one or more segments generated locally. Based on the number of segments each mapper predicts, we designed a coarse grained mapper and a fine grained mapper as shown in Figure 16. When using the framework, one can choose which type of mapper to use.

With the coarse grained mapper, each mapper explores one branch of the chunking tree, chunks the sequence into a set of segments according to the parameter, and predicts all the segments it generated locally in order. With the fine grained mapper, each mapper explores one branch of the chunking tree, chunks the sequence into a set of segments according to the parameter values and then, after the chunking finishes, the mapper only predicts one segment it generates locally. This means that if the sequence is cut into e.g., 5 segments according to the set of parameter values, there will be 5 mappers explore the same branch of the chunking tree, replicating the chunking process but predicting only one distinguished segment of the 5 chunks available. The mappers determine which segment to predict based on a hash function; thus the mappers do not need to synchronize their work or directly agree on what chunk to predict. The hash function uses the ASCII value of the chunk identifier as the key and the identifier of each mapper as the value. The function selects the segments to mappers in a round robin fashion.
3.6 Performance Analysis

The framework produces exactly the same predicted structures as the sequential code. As the prediction accuracy has already been analyzed in chapter 2, we focus only the efficiency performance of the MR framework. We use a dataset of longer sequences (i.e., 7 RNA sequences from the virus family Nodaviridae) for which the secondary structure is not known but, because of their large number of bases, the use of the MapReduce framework is vital for the efficient extensive exploration of the tree patterns. We consider the three different chunking methods - i.e., centered (C), optimized (O), and regular (R). We also consider a wide range of parameter settings - i.e., maximum chunk-length c (from 60 to 300 bases), minimum stem lengths l (from 3 to 8), and maximum gap size G (from 0 to 8).

<table>
<thead>
<tr>
<th>Sequence Name</th>
<th>Length</th>
<th>Sequence Name</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbv2</td>
<td>1399</td>
<td>nov2</td>
<td>1336</td>
</tr>
</tbody>
</table>
Table 7 Names and the lengths of the 7 longest sequences from the virus family Nodaviridae which are used to test the performance of the MapReduce framework.

<table>
<thead>
<tr>
<th>Name</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>bov2</td>
<td>1305</td>
</tr>
<tr>
<td>pav2</td>
<td>1311</td>
</tr>
<tr>
<td>fhv2</td>
<td>1400</td>
</tr>
<tr>
<td>sjnnv2</td>
<td>1410</td>
</tr>
<tr>
<td>ggnnv2</td>
<td>1433</td>
</tr>
</tbody>
</table>

We ran the MapReduce framework on a cluster composed of 6 dual quad-core compute nodes (48 cores), each with two Intel Xeon 2.50 GHz quad-core processors. A front-end node is connected to the computer nodes and is used for compilation and job submissions. A high-speed double data rate (DDR) Infiniband interconnects for application and I/O traffic and a Gigabit Ethernet interconnect for management traffic connects the compute and front-end nodes. Our implementation is based on Hadoop 0.20.2.

There are three important questions that we want to answer when measuring performance. First, we want to quantify the time spent for exploring the several branches of the search trees for the seven sequences using either centered or optimized methods, and either coarse or fine grained mapping. Second, we want to identify how the time is spent for each search in terms of map, reduce, and I/O times as well as, for the map time, in terms of compute time (for chunking and predictions) and idle time. Third, we want to measure the search efficiently and look for any aspect of the search that can impact the metric. For the measurements, we use a smaller dataset of longer sequences from the virus family Nodaviridae in Table 7.

To answer the first question, we run the 7 sequences using centered and optimized chunking methods using both coarse and fine grained mappers. When using coarse grained mappers, each mapper predicts the secondary structures of all the chunks it generates locally. When using fine grained mappers, each mapper predicts only one of the chunks it generates locally. For the latter, given a set of parameters, the chunking is replicated, i.e., the same chunking computation is performed for each chunk of a sequence. Figure 18 shows the run
time with \( c = 60, 150, \) and 300 bases. There are 12 groups on the x-axis, each group represents a configuration of mappers for the 7 sequences.

Within each group, there are 7 bars, each bar recording the execution time of one MR job for one specific sequence, one specific chunking method, and one specific maximum chunk length. On top of each bar, we record the number of map tasks run in the associated MR job. For example, the first group represents the running time of the framework when using coarse grained mapper on centered chunking method with maximum length equals 60, the second group represent the running time when using fine-grained mapper on centered chunking method with maximum length equals 60. When considering coarse grained mappers, the number of mappers is given by the combinations of \( l \) and \( G \) values (i.e., 54) while for fine grained mappers, the number of chunks equals the number of mappers and depend on the number of inversions identified in the chunking process. By comparing centered vs. optimized chunking method, we observe that the centered method results in shorter execution times, as its chunking algorithm runs faster than for the optimized method.

By comparing coarse vs. fine grained mappers, we observe that coarse grained mappers result in shorter execution time compared to fine grained mappers. We have also observed that the trend that when the maximum chunk length grows from 60 to 300, the speedup of coarse grained mappers over fine grained mapper decreases. Here we defined speedup as the ratio of the total execution time of the coarse grained mapper over the fine grained mapper as the maximum chunk length increases. We summarize the speedup of coarse grained mappers over fine grained mapper in Table 8. These results could change for a different dataset or prediction code.

<table>
<thead>
<tr>
<th>Maximum length</th>
<th>60</th>
<th>150</th>
<th>300</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td>71.8%</td>
<td>53.1%</td>
<td>26.7%</td>
<td>53.5%</td>
</tr>
</tbody>
</table>

Table 8 speedup of coarse-grained mappers over fine-grained mappers
Mapping clearly dominates the MR job time. Figure 19 cuts down the mapping time into compute time (i.e., chunking and predictions) and idle time (i.e., waiting for other mappers to complete their predictions). The figure shows how fine grained mapping reaches better efficiency compared to coarse grained mapping. In other words, with fine grained, the mappers spend more time doing real chunking and predictions. Note that the efficiency depends on the number of map tasks and the length of map tasks. The reason why fine mapping has better efficiency is these mappers have a larger number of map tasks and each map task is shorter (it predicts only one chunk). On the other hand, coarse grained mappers have a smaller number of map tasks and each map task is longer (all the sequence chunks are predicted by the mapper). We can observe these phenomena in two cases at the extreme spectrum in Figure 19, e.g., for the sequence bbv2 when using (a) the optimized chunking and a max chunk length of 300 bases for the coarse grained chunking - i.e., bad efficiency - and (b) the fine grained and the same max chunk length - i.e., good efficiency (Figure 19).

The figure shows how coarse grained mapping results in larger idle times. Similar behaviours were observed for other sequences. For the same type of mapping (fine or coarse), the centered method has better map efficiency, i.e., it spends more time chunking and predicting. The reason is that with the same maximum chunk length, the centered method tends to cut the sequence into more chunks, with shorter length than the optimized method. More chunks result in a larger number of map tasks in fine mapper while a shorter length of chunks results in shorter map tasks in both fine and coarse mappers.

To support this observation, in Figure 18 we show the number chunks and their lengths (in terms of max, min and median) with centered and optimized methods for the bbv2 sequence with 60 and 300 max chunk lengths. As shown in the figure, when the maximum length grows from 60 to 300, the number of resulting chunks for each combination of $I$ and $G$ parameters decreases, and the length of each set of chunks increases. Note that for some
combination of \( l \) and \( G \), the chunking process does not identify any set of chunks and thus for these cases we do not report any result. The overall results suggest that the best set of parameter values to achieve higher accuracy, performance, and efficiency depend on multiple aspects including the targeted sequence and the available resources. Driven by these two aspects, in future work we will integrate an automatic selection of these values into our MR framework.
Figure 18 Running time in seconds for coarse vs. fine grained mappers and centered vs. optimized methods.
Figure 19 Percentage of compute and idle time in map function for coarse vs. fine grained mappers and centered vs. optimized methods.
CHAPTER FOUR

CONCLUSIONS AND FUTURE WORK

4.1 Conclusions from work done to date

The results in the previous chapters demonstrate that sequence segmentation is a reasonable approach for circumventing the computational demands of secondary structure prediction for large RNA molecules, yet retaining a high percentage of accuracy for a prediction algorithm such as PKnotsRG. Even with the simplistic regular cutting method, over 90% of prediction accuracy is retained and both the centered and optimized cutting methods outperform the naïve regular segmentation. These primarily results support our claim that segmentation is a promising approach for predicting secondary structures of long RNA sequences, and choosing the cut points intelligently by inversion excursions can further enhance prediction accuracy.

A somewhat surprising result of our analysis is that, for the majority of our test sequences, the PKnotsRG algorithm, used with the centered and optimized cutting methods and suitable \((l, G)\) parameters, can produce even more accurate predicted secondary structures than it does without sequence segmentation. This suggests that local structures formed by pairings among nucleotides in close proximity, rather than the global thermodynamic stability of the whole molecule, may correlate better with the real structure of RNA.

We have also demonstrated that Hadoop MR is a powerful framework for describing and running various workflows in parallel, thus allowing us to gain insights on multiple modules of the RNA secondary structure prediction process easily. For example, our comparison among the MAR of the different chunking methods would have taken much longer time to complete without parallelization. It was observed that the coarse grained mappers in MR result in shorter execution time compared to fine grained mappers, although
the speedup decreases with increasing chunk lengths. We expect that using the MR framework with the coarse grained mappers will be sufficient for us to conduct a larger scale study on the impacts of different cutting methods and parameters on the accuracy retention for various secondary structure prediction algorithms, as described below.

4.2 Further Investigations

Continuing with our sequence segmentation approach, we will continue to develop the framework for a comprehensive tool for analyzing secondary structure of large RNA molecules, which was not attainable previously by existing prediction programs due to their tremendous demands on computer memory and time. Particularly for pseudoknot structure prediction, which is still challenging most computational algorithms in terms of prediction accuracy, we aim at deriving sequence specific criteria for choosing suitable inversion parameters to maximize the accuracy of the predicted structure. Specifically, I will address the following:

1. **Develop a more realistic model of inversions to reflect the nucleotide sequences in structural elements, allowing for G-U pairings and mismatches.**

In the stems of many known secondary structures from RFAM and other databases, G-U (wobble) pairings are often observed. The thermodynamic stability of G-U pairs actually approaches that of Watson–Crick pairs allows G-U pairs to substitute functionally for Watson–Crick base pairs in many RNAs [41]. Indeed, in some of our test sequences, G-U made up over 20% of the pairings in their secondary structures. Moreover, in some of the stems (especially the longer ones), a small number of mismatches are observed. In our present use of inversion distributions to determine the cut points in the segmentation process, however, neither G-U pairings nor mismatches are allowed. In order to make inversions a more realistic indicator of possible secondary structure elements, we plan to include G-U
pairings and allow mismatches in our definition of inversions. The InversFinder program that searches for inversions will be modified to accommodate these features for our future analyses on more extensive datasets and prediction algorithms (see next section). We expect that these modifications will further enhance the accuracy retention of the centered and optimized cutting methods.

2. Establish the usability of the cutting methods for various popular RNA secondary structure prediction algorithms.

In addition to the PKnotsRG algorithm, there are a number of other commonly used RNA secondary structure prediction program such as HotKnots, NUPACK, PKnotsRE, that can predict pseudoknots, and the RNAfold, UNAFOLD (previously known as mfold) that cannot. To confirm that our inversion excursion based cutting strategy can be widely used with different secondary structure prediction algorithms, we need to perform accuracy retention analysis with these popular programs. We will use an extended testing dataset comprising our current 50 testing sequences with known secondary structures, plus an additional 38 test sequences (compiled from RFAM and Pseudobase++ [44]), which are known to contain pseudoknot structures. The MapReduce framework described in Chapter 3, which executes the workflow in parallel automatically, will be vital for a thorough and efficient analysis of accuracy retention for our cutting methods on a large parameter space and the various prediction algorithms. The analysis will be performed with the MapReduce framework implemented on a cluster composed of 6 dual quad-core compute nodes (48 cores), each with two Intel Xeon 2.50 GHz quad-core processors provided by our collaborators in the Global Computing Lab at the University of Delaware.
3. **Identify criteria to determine suitable inversion parameters to attain MAR.**

Depending of the values of the minimum stem length \((l)\) and maximum gap size \((G)\), the set of inversions identified in from a nucleotide sequence will be different. In Chapter 2, we varied these two parameters, ranging from 0 and 8 for \(l\), and 3 to 8 for \(G\), and determined the \((l, G)\) combinations to give the MAR. Our results indicated that the \((l, G)\) parameter pair at which MAR is attained varies from sequence to sequence with different segmentation methods and also maximum chunk-length. With the allowance of mismatches as discussed in section 4.2.1, there will be one more parameter \(m\), namely the number of mismatches, to be considered. The selection of the best \((l, G, m)\) combination would be possible for our test sequences because their real secondary structures are known. However, when trying to make predictions for sequences with unknown secondary structure, one has to decide what would be appropriate combinations of \((l, G, m)\) for that particular sequence, prediction algorithm, and maximum chunk size. We will, therefore, need a more detailed investigation to identify criteria by which suitable \((l, G, m)\) parameters can be chosen. These criteria will guide the user when cutting the sequence, and are critical to the prediction accuracy that can be attained eventually. We anticipate that these criteria will depend on the length and composition of the sequence, as well as any knowledge of the biological characteristics of the RNA.

In the process of identifying the appropriate criteria to determine the best parameter combinations, we expect that very extensive searches on \((l, G, m)\) parameter space will be necessary. The workflow for these searches process can again fit into the Hadoop MR paradigm implemented on the University of Delaware cluster. Once these criteria are set, they can be incorporated as a decision tree into the code of our segmentation methods.
4.3 Final Remarks

Despite the promising results of our sequence segmentation approach in retaining secondary structure prediction accuracy, it is clear that any secondary structures formed by the interactions of nucleotides strings contained in different chunks will be missed by this approach. It is important to eventually extend the current segmentation process to facilitate prediction of secondary structures formed by long-range interactions of the nucleotides. Specifically, one can search for inversions across pairs of different chunks, to see if more pairings leading to more favorable secondary structures can be formed.

Our RNA secondary structure prediction tool is developed with the aim of assisting the analysis of the RNA genomes of the nodaviruses, which comprise two pieces of RNA of lengths around 1.3 and 3.2 kilobases. These viruses are of agricultural importance as they infect insects and fish. At the same time, these applications will help us fine-tune our implementation of the computational algorithms, to efficiently and accurately analyse even larger viral genomes such as those of the flaviviruses (e.g., West Nile virus, ~12 kilobases) and coronaviruses (e.g., the SARS virus, ~29 kilobases) that are of serious biomedical concerns.
A- R Code for Segmenting RNA Sequences

```perl
#!/usr/bin/perl -w

open file123, "<file.txt";
while ($lane = <file123>)
{
  chomp $lane;
  $a = $lane;

  $trial = substr $a, 13, 22;

  open percent, ">percentinput.txt";
  print percent "$trial
n";

  $trial1 = "predicted_";
  # $trial2 = "percentages_";
  $a1 = $trial1.$trial;
  # $a2 = $trial2.$trial;
  print "\n$a1\n";
  open (OUTPUT, ">$a1");
  # open (OUTPUT1, ">$a2");

  $rfamss = ' ';
  open (FILE, "$a");
  open (RFAM, "RFAM_seq.txt");
  @rfam = <RFAM>;
  close (RFAM);
  @file = <FILE>;
  close (FILE);

  # we get the file names and the rna sequence id from file
  @rnaid = ''; 
  @types = ''; 
  for ($i = 0; $i < scalar (@file); $i++)
  {
    if ($file[$i] =~ m/^[\d]+s(.*)/)
    {
      $types[$i] = $1;
      #print $1,"\n\n";
    }

    if ($file[$i] =~ m/^[\d]+s(\w+)/)
    {
      $rnaid[$i] = $1;
      #print "$1\n\n";
    }
  }

  for ($j = 0; $j < scalar (@types); $j++)
  {
    # get the sequence from rfam
    for ($h = 0; $h < scalar @rfam; $h++)
    {
      if ( $rfam[$h] =~ m/^>\s$rnaid[$j]/) # change here
      {
        #print "$rfam[$h],-m/^>\s$rnaid[$j];
        $rfamss = $rfam[$h+2];
      }
    }
  }
```

Appendix
$rfamse = $rfam[$h+1];
    #print "$rfamss\n\n";
}
}

$rfamss =~ s/[
\s]/g;
# print "$rfamss\n\n"
$lng = length $rfamss;
# print "$rfamss, $lng, $j, \n\n"
@chunk = '';
$cou = 0;
$chunkss = '';
$whole = '';
open (WHOLE, ">rna_whole.txt");
    print WHOLE $rfamse;
    close(WHOLE);
$whole =~ s/^\(\-.*//g;
$whole =~ s/\(0.00//g;
$whole =~ s/\[AUGC]///g;
$whole =~ s/\[\n\s\]///g;
$whole =~ s/{/(/g;
$whole =~ s/}/)/g;
$whole =~ s/\.://g;
# print "$whole\n\n"
for($m = 0; $m < length $rfamse; $m = $m+100)
{
    $sub = substr($rfamse, $m, 100);
    open (BITS, ">rna_100small.txt");
        print BITS "$sub\n\n";
    close(BITS);
$chunk100 =~ s/^\(\-.*//g;
$chunk100 =~ s/\(0.00//g;
$chunk100 =~ s/\[AUGC]///g;
$chunk100 =~ s/\[\n\s\]///g;
$chunk100 =~ s/{/(/g;
$chunk100 =~ s/}/)/g;
$chunk100 =~ s/\.://g;
$chunk[$cou] = $chunk100;
$chunkss = $chunkss.$chunk100;
    $cou++;
    # print "$chunkss\n";
# print length $chunkss, " c100 \n\n";

$count = 0;
@star = '';
@stop = '';
@seq = '';
@out = '';
close(RNA);

for($n=0;$n< scalar @RNA; $n++)
{
    if( $RNA[$n] =~m/^d\s(d+)\s(d+)\s(w+)/)
    { #print "$1\tt\t$2\n"
        #print $RNA[$n]; #print "$3\n"
        #print "$3\n"
        $start[$count]=$1;
        $stop[$count]=$2;
        $seq[$count]=$3;
        open (NEW ,">rna_small.txt");
        print NEW $3;
        #print $3;
        close(NEW);
        $output=`pknotsRG -k 0 -F rna_small.txt`;
        #print "$output\n"
        $output =~ s/\(+-.*\)/g;
        $output =~ s/\([AUGC]\)/g;
        $output =~ s/\[[n\s]\]/g;
        $output =~ s/\(//g;
        $output =~ s/\)/)/g;
        $output =~ s/\./:/g;
        $out[$count]=$output;
        #$xyz= length $output;
        open (ABC,">ab.txt");
        #print ABC "$output\n$xyz\n\n";
        #print "$xyz\n\n";
        $count++;
    }
}

$full='';
$mid=':';
#print :mid;
$k=1;
for($p=0; $p<$count; $p++)
{
    #print $start[$p];
    while($k<$start[$p])
    {
        $full=$full.$mid;
        $k++;
    }
    $full=$full.$out[$p];
    #print "$full";
    $k=$stop[$p]+1;
}

while($k<=$lng)
{
    $full=$full.$mid;
}
$k++;
} else{
    $full=~s/\[\n\s\]/\n/g;
    #print length $full," chunks \n";
    #print $lng," rfamss\n\n";
    if($full!~ m//(){
        #print "Error the file $types[$j] does not exist \n";
    }
    if($lng != length $full)
    {
        print "$types[$j] \n";
    }

    print OUTPUT "$types[$j]","\n";
    print OUTPUT "CHUN=$full","\n";
    print OUTPUT "RFAM=$rfamss","\n";
    print OUTPUT "WHOL=$wholess","\n";
    print OUTPUT "C100=$chunkss","\n";
    print OUTPUT "\n\n";
}
close (ABC);

}
close (OUTPUT);

###########################################################################
###############################
open(OUT,">$trial");
open(FILE1, "$a1");

@file=<FILE1>;
close(FILE1);

#we get the file names and the rna sequence id from file
@rnaid='';
@types='';
@optimzed='';
@regular='';
@centered='';
@other='';
$km=0;
$j=0;
$ko=0;
$kc=0;
$kr=0;

for($i=0; $i<scalar(@file); $i++)
{
    if($file[$i]=~ m/^RFAM/){
        $RFAM=$file[$i];
    } else{
        $RFAM=$file[$i];
    }

    $km++;
} $full=~s/\[\n\s\]/\n/g;
    #print length $full," chunks \n";
    #print $lng," rfamss\n\n";
    if($full!~ m//(){
        #print "Error the file $types[$j] does not exist \n";
    }
    if($lng != length $full)
    {
        print "$types[$j] \n";
    }

    print OUTPUT "$types[$j]","\n";
    print OUTPUT "CHUN=$full","\n";
    print OUTPUT "RFAM=$rfamss","\n";
    print OUTPUT "WHOL=$wholess","\n";
    print OUTPUT "C100=$chunkss","\n";
    print OUTPUT "\n\n";
}
close (ABC);

}
if($file[$i] =~ m/^WHO\n/){
    $WHOL = $file[$i];
}

if($file[$i] =~ m/^C100/){
    $C100 = $file[$i];
}

if($file[$i] =~ m/^\w+.bpseq\(L\dG\d)\_seg_\((\w+)\)/){
    $types[$j] = $1;
    #print $2, "\n";
    if($2 eq "Optimized_Method")
        {$file[$i+1] =~ s/CHUN=/$types[$j]O/;
         $optimized[$ko] = $file[$i+1];
         #print OUT $file[$i+1];
         $ko ++;
        }
    elsif($2 eq "Centered_Method")
        {$file[$i+1] =~ s/CHUN=/$types[$j]C/;
         $centered[$kc] = $file[$i+1];
         #print OUT $file[$i+1];
         $kc ++;
        }
    elsif($2 eq "Regular_Method")
        {$file[$i+1] =~ s/CHUN=/$types[$j]R/;
         $regular[$kr] = $file[$i+1];
         #print $file[$i];
         $kr ++;
        }
    else{
        $file[$i+1] =~ s/CHUN=/$types[$j]M/;
        $other[$km] = $file[$i+1];
        #print $file[$i+1];
        $km ++;
    }
}

if($file[$i] =~ m/^\w+\./){
    $rnaid[$j] = $1;
    #print $1, "\n";
}

$j ++;

#print $ko, $kc, $kr, $km;

for($l = 0; $l < scalar (@optimized); $l ++){
    print OUT $RFAM;
    print OUT $WHOL;
    print OUT $optimized[$l];
    print OUT $centered[$l];

print OUT $C100;
print OUT "\n\n";

print $RFAM;
print $WHOL;
print $optimized[$l];
print $centered[$l];
print $C100;
}

for($l1=0; $l1< scalar (@other); $l1++){
print OUT $other[$l1];
}

###########################################################################
############
}

###########################################################################
############
}
B- Perl Code for Percentage comparison

```perl
open percent, "<percentinput.txt"
while($line123=<percent>)
{
    chomp $line123;

    open NEWDATA, "<$line123"
    $result12=RESULT;
    $result123=$result12.$line123;
    @a=" ";@b=" ";
    open RESULT, ">$result123"
    @i=-1;$j=0;
    $seq=seq;
    $seq123=$seq.$line123;
    open seq, ">$seq123"
    while($line=<NEWDATA>)
    {
        chomp $line;
        $x1234= substr $line, 0, 4;
        if($x1234 eq "RFAM")
        {
            $i+=2;$j=0;
        }
        $j++;
        $a[$i][$j] = substr $line, 5;
        if($x1234 eq "RFAM")
        {
            for($a1=1;$a1<=$i;$a1+=2)
            {
                for($b1=1;$b1<=5;$b1++)
                {
                    if($b1 eq 1)
                    {
                        @l = split(undef,$a[$a1][$b1]);
                        for $l(@l)
                        { $b[$a1][$b1][$p] = $l;
                            print RESULT "$p\t$b[$a1][1][$p]\n";$p++;}
                    }
                    if($b1 eq 2)
                    {
                        @m = split(undef,$a[$a1][$b1]);
                        for $m(@m)
                        { $b[$a1][$b1][$q] = $m;
                            print RESULT "$q\t$b[$a1][2][$q]\n";$q++;}
                    }
                    if($b1 eq 3)
                    {
                        @nx = split(undef,$a[$a1][$b1]);
                        for $nx(@nx);
```
{
$b[a1][b1][r]= nx;
print RESULT "$r\t$b[a1][b1][r]\n";
$r++;})
if($b1 eq 4)
{s=1;
# print seq "$a[$a1][b1]\n";
@p = split(undef,$a[$a1][b1]);
for $p(@p)
{$b[$a1][b1][$s]= $p;
# print seq "$a\t$b1\ts\t$s\n";
print RESULT "$s\t$b1[$s]\n";$s++;}}
if($b1 eq 5)
{
$pr=1;
@l = split(undef,$a[$a1][b1]);
for $l(@l)
{$b[$a1][b1][$pr]= $l;
print RESULT "$pr\t$b1[$pr]\n";$pr++;}}
}
}
close(RESULT);
for($a=1;$a<=$i;$a+=2)
{
for($k=1;$k<=5;$k++)
{# print seq "$a[$a][k]\n";
for($x=1;$x<=$p;$x++)
{
# print seq "$a\t$k\t$x\t$b[a][k][x]\n";
})
}$p--;
$r12=R_;
$r123=$r12.$line123;
print "\n$r123\n";
open R, ">$r123";
for($a=1;$a<=$i;$a+=2)
{
for($k=1;$k<=5;$k++)
{
for($x=1;$x<=$p;$x++)
{
if($b[$a][k][x] eq '(')
{
print R "$x\t$b[a][k][x]\n";
}
if($b[$a][k][x] eq ')')
{
print R "$x\t$b[a][k][x]\n";
}
open RESULT, "<$result123";
for($x=1;$x<=$p;$x++)
{
if($b[$a][$k][$x] eq ':')
{
    print R "$x\t$b[$a][$k][$x]\n";
}
print R \n;  
print "$a\t$k\t"; 
}

close(NEWDATA);
close(RERESULT);
close(R);

close(ND);
open ND, "<$r123";
$res12=RES;
$res123=$res12.$line123;
open RES, ">$res123";
$i=0;
while($line=<ND>)
{
    chomp $line;
    ($d,$e,) = split("\t",$line);
    if($e eq '(')
    {
        $i++;
        $a[$i]=$d,
    } 
    if($e eq ')')
    {print RES "$d\t$a[$i]\n";
        $i--;
    }
    if($e eq ':')
    {print RES "$d\t$d\n";
    } if($d eq "" && $e eq ""){print RES "\n";
    }
}
close(ND);
close(RES);

@jk=" ";
$abd12=abd_; 
$abd123=$abd12.$line123;
open RES30, ">$abd123";
open RES2, "<$r123";
$i=0;
while($lane=<RES2>)
{
    chomp $lane;
    ($d,$e,) = split("\t",$lane);
    $jk[$i]=$d;
    $i++;
}
@gh = reverse sort { $a <=> $b } @jk;

open RES1, "<$res123";
$m=1;$ad=1;$i=1;$x=0;$u=0;$s=0;$n=0;
while($lane=<RES1>)
{
  chomp $lane;
  ($d,$e,) = split("\t",$lane);
  $b[$m][$i]=$lane;

  $y[$ad]=$i;
  $ad++; $i++;

  if($d eq "" && $e eq "")
  {
    $i--;
    #print "$m,$i\n";
    $m++; $i=1;
  }
}

$j=$i+1;

$ik=1;

$pl23="percent _";
$pl24=$pl23.$line123;
open Z, ">$p124";
open RES3, "<$line123";
while($lune=<RES3>)
{
  $ghk[$ik]=$lune;
  $ik++;
}

for($ik1=3;$ik1<$ik;$ik1+=7)
{
  ($d1,$e1,) = split("O",$ghk[$ik1]);
  $ghk1[$ik1]=$d1;
}

print Z "\tRFAM/L3G0c\tRFAM/L3G0c\tRFAM/C100\tRFAM/WHOL\tWHOL/L3G0c\tWHOL/C100\tACR/AW\tACC/AW\tACO/AW\n"

@g = reverse sort { $a <=> $b } @y;

$qw=3;$qwl=0;$fk=3;
for($k=1;$k<$m;$k++)
{
  for($j=1;$j<=$g[0];$j++)
  {
    for($f=1;$f<=$g[0];$f++)
    {
      ($q, $w,) = split("\t",$b[$k][$j]);
      ($ql1, $wl1,) = split("\t",$b[$k+1][$j]);
      ($x, $y,) = split("\t",$b[$k+1][$f]);
      ($r, $s,) = split("\t",$b[$k+2][$f]);
      ($u, $v,) = split("\t",$b[$k+3][$f]);
    
    ...
\begin{verbatim}

\$u1, \$v1, = \text{split}("\textbackslash t", \$b[\$k+4][\$f]);

if(\$q eq \$x && \$w eq \$y)
    \{
        \if(\$x eq "" && \$y eq "" && \$q eq "" && \$w eq "")
            \print \text{RES30} "$q eq \$x && \$w eq \$y\textbackslash n$\n        \else(\$q eq \$w && \$x eq \$y)
            \{$n++; \print \text{RES30} "$q eq \$x && \$w eq \$y\textbackslash n$\n        \}
    \else($n+=2; \print \text{RES30} "$q eq \$x && \$w eq \$y\textbackslash n$\n    \}

($r, \$s, ) = \text{split}("\textbackslash t", \$b[\$k+2][\$f]);
if(\$q eq \$r && \$w eq \$s)
    \{
        \if(\$r eq "" && \$s eq "" && \$q eq "" && \$w eq "")
            \{}
        \elsif(\$q eq \$w && \$r eq \$s)
            \{$n1++; \}
        \else($n1+=2; \}
    \}

($u, \$v, ) = \text{split}("\textbackslash t", \$b[\$k+3][\$f]);
if(\$q eq \$u && \$w eq \$v)
    \{
        \if(\$u eq "" && \$v eq "" && \$q eq "" && \$w eq "")\{}
        \elsif(\$q eq \$w && \$u eq \$v)
            \{$n2++; \}
        \else($n2+=2; \}
    \}

($u1, \$v1, ) = \text{split}("\textbackslash t", \$b[\$k+4][\$f]);
if(\$q eq \$u1 && \$w eq \$v1)
    \{
        \if(\$u1 eq "" && \$v1 eq "" && \$q eq "" && \$w eq "")\{}
        \elsif(\$q eq \$w && \$u1 eq \$v1)
            \{$n3++; \}
        \else($n3+=2; \}
    \}

(\$q1, \$w1, ) = \text{split}("\textbackslash t", \$b[\$k+1][\$j]);
($r, \$s, ) = \text{split}("\textbackslash t", \$b[\$k+2][\$f]);
if(\$q1 eq \$r && \$w1 eq \$s)
    \{
        \if(\$r eq "" && \$s eq "" && \$q1 eq "" && \$w1 eq "")\{}
        \elsif(\$q1 eq \$w1 && \$r eq \$s)
            \{$n11++; \}
        \else($n11+=2; \}
    \}

($u, \$v, ) = \text{split}("\textbackslash t", \$b[\$k+3][\$f]);
if(\$q1 eq \$u && \$w1 eq \$v)
    \{
        \if(\$u eq "" && \$v eq "" && \$q1 eq "" && \$w1 eq "")\{}
        \elsif(\$q1 eq \$w1 && \$u eq \$v)
    \}
\end{verbatim}
($n21++;
} else{$n21+=2;
}

($u1,$v1,)= split("\t",$b[$k+4][$f]);
if($q1 eq $u1 && $w1 eq $v1)
{
 if($u1 eq "" && $v1 eq "" && $q1 eq "" && $w1 eq ""){}
 elseif($q1 eq $w1 && $u1 eq $v1)
 {($n31++;
} else{($n31+=2;
}

$k+=4;
$xyz=$gh[0];
$n=sprintf("%4.4f",$n/$xyz);$n1=sprintf("%4.4f",$n1/$xyz);$n2=sprintf("%4.4f",$n2/$xyz);$n3=sprintf("%4.4f",$n3/$xyz);$n11=sprintf("%4.4f",$n11/$xyz);$n21=sprintf("%4.4f",$n21/$xyz);$n31=sprintf("%4.4f",$n31/$xyz);$xy1=sprintf("%4.4f",$n1/$n);$xy2=sprintf("%4.4f",$n2/$n);$xy3=sprintf("%4.4f",$n3/$n);

if($qwl eq 9){$qw++;$qwl=1;}

print Z "$ghk1[$fk]\t$n1\t$n2\t$n3\t$n\t$n11\t$n21\t$n31\t$xy1\t$xy3\t$xy2\t$xy1"
$n=0;$n1=0;$n2=0;$n3=0;$n11=0;$n21=0;$n31=0;$xy1=0;$xy2=0;$xy3=0;
print Z "\n"
$fk+=7;
}

close(Z);
close(RES1);
close(RES30);
close(RES2);
#!usr/bin/perl -w

open file123, "<file.txt";
while($lane=<file123>)
{
    chomp $lane;
    $a=$lane;

    $trial=substr $a, 13, 22;

    open percent,">percentinput.txt";
    print percent "$trial
"

    $trial1="predicted_";
    $trial2="percentages _";
    $a1=$trial1.$trial;
    $a2=$trial2.$trial;
    print "\n$a1\n"
    open (OUTPUT, ">$a1");
    open (OUTPUT1, ">$a2");

    $rfamss=' ';
    open (FILE, "$a");
    open (RFAM, "RFAM_seq.txt");
    @rfam=<RFAM>;
    close (RFAM);
    @file=<FILE>;
    close (FILE);

    #we get the file names and the rna sequence id from file
    @rnaid=' ';
    @types=' ';
    for($i=0; $i< scalar (@file); $i++)
    {
        if($file[$i]=~ m/^[\d]+s(.*)/){
            $types[$i]=$1;
            # print $1, "\n"
        }

        if($file[$i]=~ m/^[\d]+s(\w+)/){
            $rnaid[$i]=$1;
            # print "$1\n"
        }
    }

    for($j=0;$j< scalar (@types); $j++){
        # get the sequence from rfam
        for($h=0;$h< scalar @rfam; $h++)
        {
            if( $rfam[$h] =~ m/^[^>\s]$rnaid[$j]/)# change here
            {
                #print "$rfam[$h], $rnaid[$j]"
                $rfamss= $rfam[$h+2];
                $rfamss= $rfamss;
                print "$rfamss
\n"
            }
        }
    }

C- Perl Code for Predicting the Secondary Structure of Chunks
$rfamss =~ s/\[\n\s]/\s/g;
# print "$rfamss
"
$lng = length $rfamss;
# print "$rfamss,$lng,$j,\n\n"
@chunk = ''; $cou = 0; $chunkss = ''; $wholess = '';
open (WHOLE, ">rna_whole.txt");
print WHOLE $rfamss;
close(WHOLE);
$wholess = `pknotsRG -k 0 -F rna_whole.txt`;
$wholess =~ s/\(.*//g;
$wholess =~ s/\(0.00\)///g;
$wholess =~ s/\[AUGC]///g;
$wholess =~ s/\[\n\s]/\s/g;
$wholess =~ s/\{/\(/g;
$wholess =~ s/\}/\)/g;
# print "$wholess
\n"
for ($m = 0; $m < length $rfamss; $m = $m + 100) {
$sub = substr($rfamss, $m, 100);
open (BITS, ">rna_100small.txt");
print BITS "$sub
"
close(BITS);
$chunk100 = `pknotsRG -k 0 -F rna_100small.txt`;
$chunk100 =~ s/\(.*//g;
$chunk100 =~ s/\(0.00\)///g;
$chunk100 =~ s/\[AUGC]///g;
$chunk100 =~ s/\[\n\s]/\s/g;
$chunk100 =~ s/\{/\(/g;
$chunk100 =~ s/\}/\)/g;
$chunk[$cou] = $chunk100;
$chunkss = $chunkss.$chunk100;
$cou++;
}
# print "$chunkss
"
# print length $chunkss, " c100 \n"

$count = 0;
@start = '';
@stop = '';
@seq = '';
@out = ''; 

open (RNA, "$types[$j]");
@RNA = <RNA>;
close(RNA);

for ($n = 0; $n < scalar @RNA; $n++)
if( $RNA[$n] =~m/^d(s(d+)(d+)(w+))/ )
{
  #print "$1\t\t$2\n\n";
  #print $RNA[$n];
  #print "$3\n";
  $start[$count]=$1;
  $stop[$count]=$2;
  $seq[$count]=$3;
  open (NEW , "rna_small.txt");
  print NEW $3;
  #print $3;
  close(NEW);
  $output=`pknotsRG -k 0 -F rna_small.txt`;
  #print "$output\n\n";
  $output=~ s/\([-\.]\+//g;
  $output=~ s/\((0.00)\+//g;
  $output=~ s/[AUGC]//g;
  $output=~ s/[\n\s]//g;
  $output=~ s/./:/g;
  $output=~ s/{/(/g;
  $output=~ s/}/)/g;
  $output=~ s/\/./\.:/g;

  $out[$count]=$output;
  #$xyz= length $output;
  open (ABC, ">ab.txt");
  #print ABC "$output\n $xyz\n\n";
  #print "$xyz\n\n";
  $count++;
}

$full='';
$mid=':';
#print $mid;
$k=1;
for($p=0; $p<$count; $p++)
{
  #print $start[$j];
  while($k<$start[$p])
  {
    $full=$full.$mid;
    $k++;
  }
  $full=$full.$out[$p];
  #print "$full";
  $k=$stop[$p]+1;
}

while($k<=$lng)
{
  $full=$full.$mid;
  $k++;
}
$full=~s/\[\n\s]//g;
#print length $full, " chunks \n";
#print $lng, " rfamss\n\n"
if($full!~ m/\(/)
{
#print "Error the file $types[$j] does not exist \n";
}
if($lng != length $full)
{
print " $types[$j] \n";
}
print OUTPUT "$types[$j]","\n"
print OUTPUT "CHUN=$full","\n"
print OUTPUT "RFAM=$rfamss","\n"
print OUTPUT "WHOL=$wholess","\n"
print OUTPUT "C100=$chunkss","\n"
print OUTPUT "\n\n"
close (ABC);

}
close (OUTPUT);

########################################################################
#we get the file names and the rna sequence id from file
@file=<>;
@rnaid='';
@types='';
@optmized='';
@regular='';
@centered='';
@other='';
$km=0;
$j=0;
$ko=0;
$kc=0;
$kr=0;
for($i=0; $i<scalar(@file); $i++)
{
   if($file[$i]=~ m/^RFAM/){
       $RFAM=$file[$i];
   }
   if($file[$i]=~ m/^WHOL/){
       $WHOL=$file[$i];
   }
}
if($file[$i] =~ m/^C100/){
    $C100=$file[$i];
}

if($file[$i] =~ m/^\w+\bpseq\(L\dG\d)\_seg\_(\w+)\.*$/){
    $types[$j]=$1;
    #print $2,"\n";
    if($2 eq "Optimized_Method"){
        $file[$i+1] =~ s/CHUN=/types[$j]O/;
        $optimized[$ko]=$file[$i+1];
        #print OUT $file[$i+1];
        $ko++;
    }
    elsif($2 eq "Centered_Method"){
        $file[$i+1] =~ s/CHUN=/types[$j]C/;
        $centered[$kc]=$file[$i+1];
        #print OUT $file[$i+1];
        $kc++;
    }
    elsif($2 eq "Regular_Method"){
        $file[$i+1] =~ s/CHUN=/types[$j]R/;
        $regular[$kr]=$file[$i+1];
        #print $file[$i];
        $kr++;
    }
    else{
        $file[$i+1] =~ s/CHUN=/types[$j]M/;
        $other[$km]=$file[$i+1];
        #print $file[$i+1];
        $km++;
    }
}

if($file[$i] =~ m/^\(\w+)\.*$/){
    $rnaid[$j]=$1;
    #print $1,"\n";
}
$j++;
}

#print $ko,$kc,$kr,$km;

for($l=0; $l < scalar (@optimized); $l++){
    print OUT $RFAM;
    print OUT $WHOL;
    print OUT $optimized[$l];
    print OUT $centered[$l];
print OUT $C100;
print OUT "\n\n";

print $RFAM;
print $WHOL;
print $optimized[$l];
print $centered[$l];
print $C100;

}

for($l1=0; $l1< scalar (@other); $l1++){ 
print OUT $other[$l1];
}$l1++;
}

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

}
References


Biography

Daniel T. Yehdego was born in Asmara, Eritrea. The son of Tesfai Yehdego and Negeset Woldu. Daniel graduated from Ibrahim Sultan Secondary School (Asmara, Eritrea), in the fall of 1999 and entered The University of Asmara in the country of Eritrea in the spring of 2000. He graduated from The University of Asmara with a B.Sc. in Physics and Education in the summer of 2004. He entered the Physics Masters program at the University of Texas at El Paso in the spring semester of 2008 and earned a M.Sc in Physics in Spring 2009. He has been a Teaching Assistant since the spring of 2008 for various classes at the UTEP physics department. Daniel is currently a graduate PhD student in the Computational Science program of the University of Texas at El Paso.

Mailing address: 76 Via Verde, San Lorenzo

California, 94580

This Thesis was typed by Daniel T. Yehdego