German University in Cairo
Faculty of Media Engineering and Technology

ExpaRNA-P

Bachelor Thesis

Author:	Noha Waheed Radwan
Supervisor:	Prof. Dr. Rolf Backofen
Co-supervisor:	Dr. Ing. Mathias Moehl
Co-supervisor:	Dipl. Bioinf. Christina Schmiedl
GUC-supervisor:	Prof. Dr. Slim Abdennadher

This is to certify that:

(i) the thesis comprimises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

Noha Waheed Radwan

Abstract

A RiboNucleic Acid (RNA) is a single stranded nucleic acid that plays an important role in the cell. An RNA sequence string is a string over the alphabet $\{A, U, C, G\}$. The function performed by a non-coding RNA is dictated by its structure. Therefore to understand more about the similarities and differences between RNA strands we need a tool that given two RNA sequences would get the maximum non-overlapping set of exact pattern matchings. Knowing about these similarities and differences would help us more in understanding the functions of the RNAs.
In this thesis, we develop a new version of the tool *ExpaRNA*, called *ExpaRNA-P* which uses a different algorithm than the one used in *ExpaRNA*; it considers the entire ensemble of structures possible to an RNA sequence in order to find the longest non-overlapping set of exact pattern matches in both sequences whereas *ExpaRNA* solves this problem but only for a fixed structure. The output from both *ExpaRNA* and *ExpaRNA-P* can later be fed into the alignment program *LocARNA* as anchor constraints speeding up the algorithm.

Contents

Chapter 1

Introduction

1.1 Motivation

A RiboNucleic Acid (RNA), is a macromolecule that is present in all living cells. Its chemical structure is very similar to that of DNA; it consists of a single chain (unlike double chain in DNA) of nucleotides. Each nucleotide is formed from a nitrogenous base, a ribose sugar and a phosphate group. In an RNA sequence, there are four different types of nitrogenous bases that exist; Adenine (A), Uracil (U), Cytosine (C) and Guanine (G). RNA plays a major role in the cell as it encodes genetic information and thus controls gene expression. Its roles also include protein synthesis; it plays an important role in the transcription and the translation process, catalyzing biological reactions and sensing and communication responses to cellular signals [15].
For any given RNA sequence, there exists more than one possible structure. Since the structure of the RNA was known to define its function and role in the cell, more interest grew into finding the similarities and differences between different RNA families and structures.

In the paper by Heyne et al. [13] *ExpaRNA* was developed, which is a tool for finding the longest set of non-overlapping exact matches in RNA sequences. The input to *ExpaRNA* is two RNA sequences and their fixed structure, and the output is the maximal non-overlapping set of maximal exact pattern matchings. This output can then be fed into the *LocARNA* tool [11] as constraints for the alignment. The benefit gained from this is a faster output from *LocARNA* as the constraints would restrict the *LocARNA* search space and thus speeding up the alignment process.
In this thesis, we use a different algorithm for calculating the exact pattern matches than the one used by Heyne et al [13] for *ExpaRNA*. Instead of assuming a fixed structure for the RNA sequence, the entire ensemble of structures is considered. The output is the best scored set of maximal non-overlapping exact pattern matchings under restriction by time and space complexity, to remain as comparable as possible to the *ExpaRNA* tool.

1.2 Aim of the project

The aim of this project is to design a tool that given two RNA sequences would output the maximal non-overlapping set of exact pattern matchings, while considering all possible structures. We combine our tool with RNA alignment tools and test it against the k2 data set from the Bralibase benchmark [2], [6] to measure the accuracy of the produced alignment in comparison to the *ExpaRNA* tool.

1.3 Thesis structure

The thesis below is further classified into four chapters as follows:

- **Background:** This chapter discusses in detail all the technologies used in the research to create the tool and any background information required to understand the novel algorithm.
- **Exparna-P:** This chapter presents all the theoretical and algorithmic details regarding the implementation of the application. It describes how to install the tool; what are the prerequisites. It also gives a walk-through for the parameters that can be set by the user, explaining each one briefly.
- **Implementation:** This chapter discusses the working environment of the tool from programming languages and repositories. It also discusses the parameters of the tool, their default values and the logic behind setting them to such values.
- **Evaluation:** This chapter contains the details of the evaluation of our tool and a comparison to other alignment tools. It has two main parts, a small scale evaluation where we do a comparative structural analysis for large RNA sequences. The second part would be a more large scale analysis which will be evaluated with the Bralibase 2.1 benchmark [2].
- **Conclusion:** This chapter contains the final discussion and summary of the whole research and how it has managed to solve the problems that were presented. It will further discuss the limitations of the project and what could be done in the future to improve the application features upon the work presented in this thesis.

Chapter 2

Background

This chapter describes in detail the theoretical background technologies required and used in the development and research of the application presented in this thesis. It starts off with some formal definitions and technical abbreviations (Section 2.1). Then in the Technical Background (Section 2.2) we discuss briefly the tools and libraries that were useful or related in a way to our work. The last section of this chapter discusses the algorithms that were used in our tool (Section 2.3).

2.1 Preliminary Definitions

We begin this section by formally defining an RNA sequence S:

Definition 2.1.1 (Sequence) *S is a string such that:*

$$\forall S[i], S[i] \in \{A, U, C, G\}$$

where $S[i]$ is the nucleotide at the i-th position of S.

Definition 2.1.2 (Structure) *A structure P of S is defined as follows:*

$$P = \{p_1, \ldots, p_k\}$$

where $p = (i, j)$ such that $1 \leq i < j \leq n$, where $S[i]$ and $S[j]$ must be complementary Watson-Crick base pairs (A-U or C-G) or a non-standard base pair G-U.

For a base pair $p = (i, j)$, its right end j can be referred to by p^R. The left end of the arc i can be referred to by p^L.
For a single structure, we assume that each sequence position is involved in at most one base pair i.e. for all $(i, j), (i', j') \in P$, $i = i' \Leftrightarrow j = j'$ and $i \neq j'$ and base pairs do not cross.

Consider two structures P_1, P_2, with arcs $p_1 = (i_1, j_1) \in S_1, p_2 = (i_2, j_2) \in S_2$.

Definition 2.1.3 (Arc match) *Arcs p_1 and p_2 are said to form an arc match if $S_1[i_1] = S_2[i_2]$ and $S_1[j_1] = S_2[j_2]$.*

Different types of loops that could exist within a RNA structure.

A stem is substructure within an RNA strand where positions $(i, j), (i+1, j-1)$etc form base pairs. Fig. 2.1 (A) shows the appearance of a stem within an RNA strand.

A Hairpin loop H is an area where an RNA strand has folded back on itself and nucleotides from the two separate segments have base paired, so that the resulting structure appears as a hairpin. A visualization for the Hairpin loop is shown in Fig. 2.1 (B).

An Interior/Internal loop I is created when an interruption in the RNA strand occurs by a series of bases that cannot form standard Watson-Crick pairs, thus resulting in a structure as the one shown in Fig. 2.1 (C).

A Bulge loop B has a similar structure to the Interior loop except that the interruption occurs in only one RNA strand; instead of in both RNA strands which is the case with the Interior loop; thus resulting in a bulge in one of the RNA strands (illustration shown in Fig. 2.1 (D)).

A Multiple loop M occurs when more than two double-stranded regions converge to form a closed structure. Fig. 2.1 (E) illustrates the multiple loop in an RNA strand.

Definition 2.1.4 (Nested Structure) *A structure is said to be nested, if all the base pairs are non-crossing, i.e. let there be arcs $p_1 = (i, j), p_2 = (i‘, j‘)$, then the structure is nested if $i < j < i‘ < j‘$ or $i < i‘ < j‘ < j$.*

An example of a nested structure is shown in fig. 2.2 (A).

Definition 2.1.5 (Crossing Structure) *A structure is said to be crossing if there exists two or more base pairs (i, j) and $(i‘, j‘)$ with $i < i‘ < j < j‘$.*

Fig. 2.2 (B) shows an example of a crossing structure of an RNA strand.

Definition 2.1.6 (Pseudo-knot) *A* pseudo-knot *occurs in an RNA sequence as a result of a crossing structure. The base pairing in a* pseudo-knot *is not well nested; that is, base pairs occur that "overlap" one another in sequence position [12].*

Fig. 2.2 (C) visualizes the appearance of a *pseudo-knot*.

Definition 2.1.7 (Exact Matching) *An exact matching between two RNA sequences S_1 and S_2 is the set $E \subseteq \{1, \ldots, |S_1|\} \times \{1, \ldots, |S_2|\}$ such that for all $(i_1, i_2) \in M$ it holds that $S_1[i_1] = S_2[i_2]$. The set of positions in an exact matching must be connected either on sequence or structural level. A matching on sequential level occurs by matching unpaired nucleotides in S_1 and S_2. A structural matching occurs by matching a base pair in S_1 and S_2. Fig. 2.3 shows structural and sequential matching for an exact matching.*

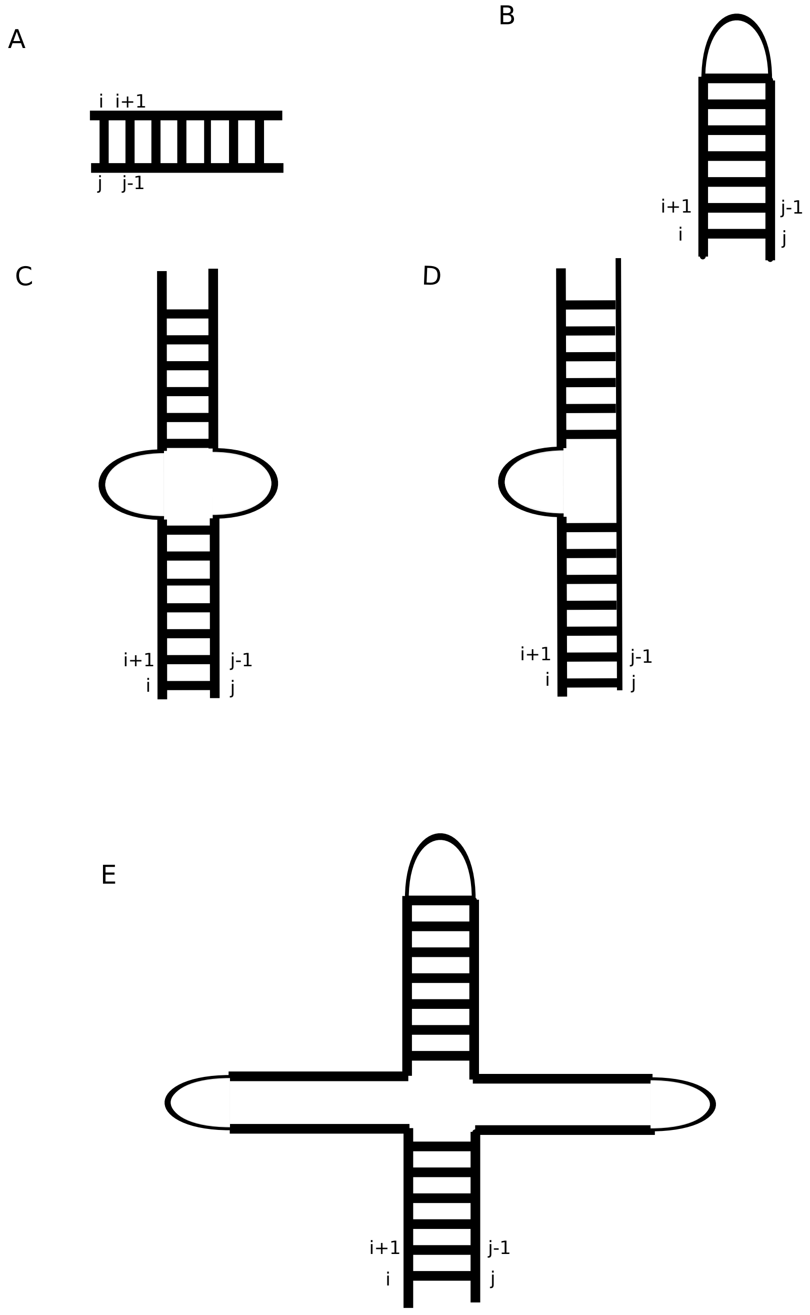

Figure 2.1: The figure illustrates the types of loops. (A) shows a stem substructure, (B) shows a hairpin loop, (C) shows the internal loop, (D) shows a bulge loop, and (E) is the multiple loop

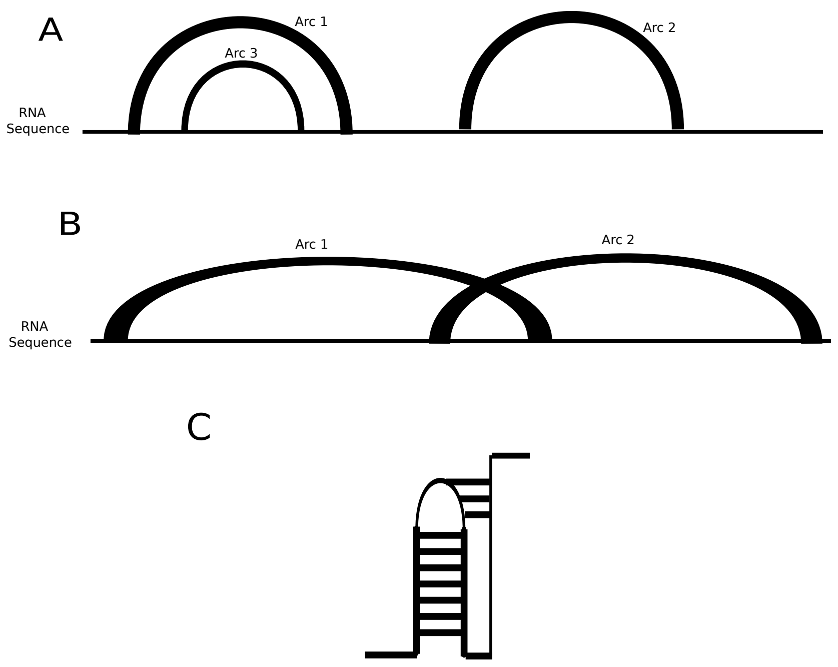

Figure 2.2: Different structures illustration. (A) Representation of nested structure. It contains three arcs. Arcs 1 and 2 satisfy the first condition in the theorem, arcs 1 and 3 satisfy the other condition. (B) Representation of crossing structure. It consists of two crossing stem intervals. (C) Corresponding pseudo-knot structure

2.2 Technical Background

This section will cover the technologies used in this thesis.

2.2.1 Vienna RNA Package

The Vienna RNA Package is a library implemented in C programming language. The version of the Vienna RNA package used during the implementation of this tool is RC2 (version 2.0.0). It contains a number of algorithms and programs for calculating, predicting and comparing RNA secondary structures. Out of the several algorithms for structure prediction included in the package, we use mainly two algorithms; the partition function algorithm and the folding algorithm. The partition function algorithm was developed by McCaskill in 1990 for computing the energy function and base pairing probabilities for all possible structures of an RNA sequence [8]. The other algorithm used when calculating the graphical output of the tool, is the folding algorithm developed by Wuchty et.al. (1999) [14]. Given an RNA sequence, it outputs the most probable structure for it within a given range of optimal energy. The package contains other algorithms than the two mentioned above. However, these are the only two of interest to us during the development of the tool.

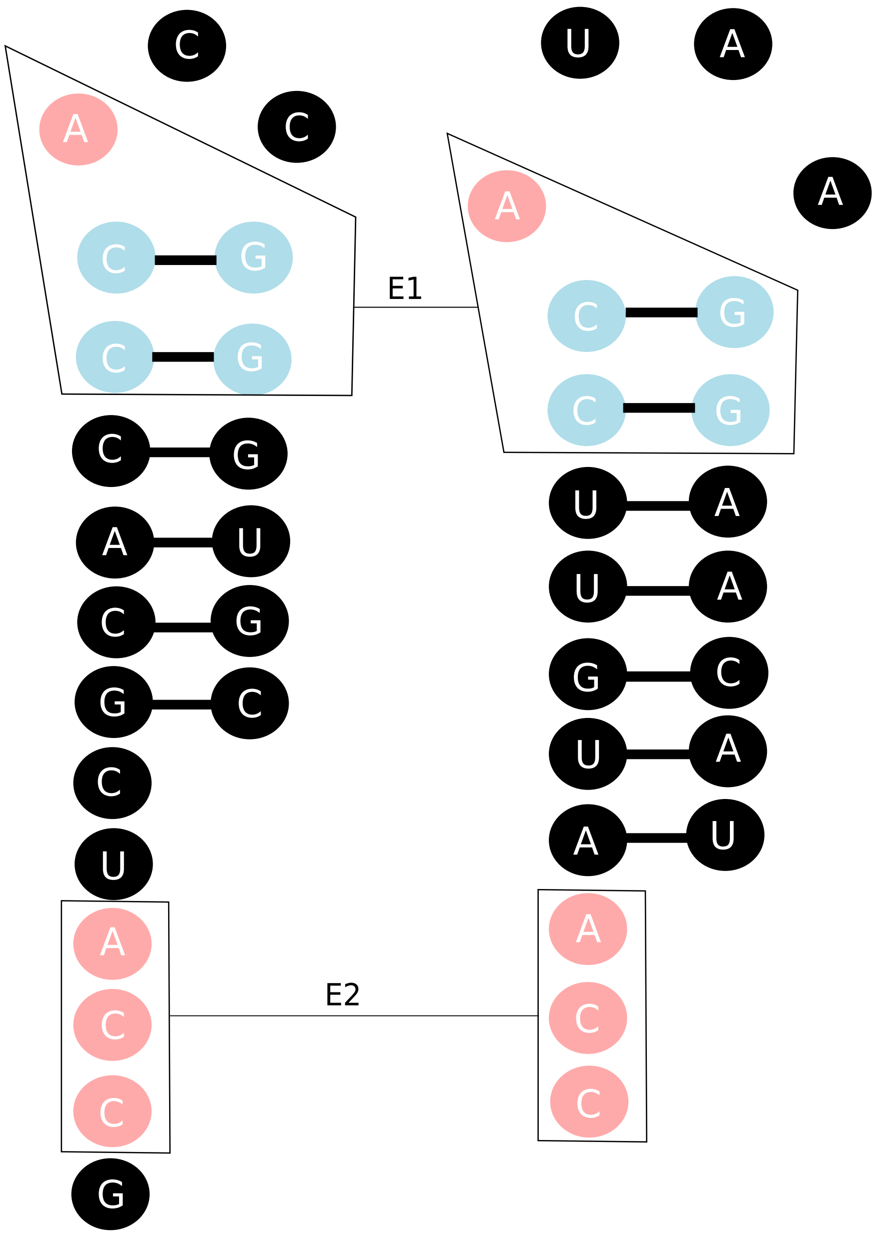

Figure 2.3: The figure above shows to exact matchings. It shows the types of matchings possible in an exact matching. The positions colored in pink are matched sequentially in both sequences, while the positions colored in blue are matched structurally. You can see from the figure that sequential and structural matchings can occur interleaved in the case of matching unpaired bases underneath an arc in E1.

2.2.2 LocARNA

LocARNA is a tool for RNA alignment. Given two RNA sequences, it performs the folding and alignment for the input sequences and produces the best alignment and consensus structure for both sequences [11].

2.2.3 ExpaRNA

ExpaRNA is a tool for structural-based comparison of RNA molecules. It requires a given structure for the input sequences. However, if no structure is given at run time, the RNAfold algorithm predicts the Minimum Free Energy (MFE) structure, which is then used during the calculations by *ExpaRNA*. *ExpaRNA* computes the best arrangement of common substructures for the input sequences that is non-crossing, non-overlapping and at the same time has the best score [13].

2.3 Algorithms

This section describes the algorithms that were used in our tool.

2.3.1 McCaskill Algorithm

The McCaskill algorithm is an algorithm to compute the partition function over all secondary structures as well as the base pairing probabilities. We will briefly explain the McCaskill algorithm according to the way it is implemented in the Vienna RNA package.
Definition 1: (Partition Function)
The partition function of an RNA molecule with energy function $E[P]$, is given by:

$$Z \equiv \sum_{S} e^{-\frac{E[S]}{RT}}$$

where

$$R = 1.987 \frac{cal}{molK}$$

, P is a structure of an RNA sequence S, and T is the temperature in Kelvin.
The term $e^{-\frac{E[S]}{RT}}$ is referred to as the Boltzmann weight for the structure P of the sequence S. The partition function is a sum of all Boltzmann factors for all possible structures P of an RNA sequence S. Once calculated, it can be used to compute the probability of any given structure for an RNA sequence.

The Vienna RNA library uses the McCaskill Algorithm as an intermediate step in its calculations.
The McCaskill algorithm implemented in version 2.0.0 of the Vienna RNA package, creates the following matrices for the set of structures of $P[i, j]$:

- Q_{ij}: sum of the Boltzmann weights of all structures of $P[i, j]$.

- Q^q_{ij}: sum of the Boltzmann weights of all non-empty structures of $P[i,j]$ that have a base pair starting at position i and is not a part of any loop (i.e. it is the outermost base pair).
- Q^b_{ij}: sum of the Boltzmann weights of all structures of $P[i,j]$ with a base pair (i,j) as an outer base pair for a hairpin loop, interior loop or multiple loop.
- Q^m_{ij}: sum of the Boltzmann weights of all non-empty structures of $P[i,j]$ which have at least one base pair and is part of a multiple loop.
- Q^{qm}_{ij}: sum of the Boltzmann weights of all non-empty structures of $P[i,j]$ which are a part of a multiple loop and have at least one base pair starting at position i.

Initially, the matrices are filled with the following values:

- $Q_{ij} = 1$
- $Q^q_{ij} = 0$
- $Q^b_{ij} = 0$
- $Q^m_{ij} = 0$
- $Q^{qm}_{ij} = 0$

The recursion used to compute these matrices is given below:

$$\begin{aligned}
Q_{ij} &= U_1(j-i+1) && (2.1)\\
&\quad + Q^q_{ij} && (2.2)\\
&\quad + \sum_{\substack{k\\ i\le k<j}} (Q_{ik}Q^q_{k+1j}) && (2.3)\\
Q^q_{ij} &= Q^q_{ij-1}U_1(1) && (2.4)\\
&\quad + Q^b_{ij}U_2(i,j) && (2.5)\\
Q^b_{ij} &= H(i,j) && (2.6)\\
&\quad + \sum_{\substack{k,l\\ i<k<l<j}} (Q^b_{kl}I(i,j,k,l)) && (2.7)\\
&\quad + \sum_{\substack{k\\ i<k<j}} (Q^m_{i+1k-1}Q^{qm}_{kj-1})aM(i,j) && (2.8)\\
Q^m_{ij} &= Q^{qm}_{ij} && (2.9)\\
&\quad + \sum_{\substack{k\\ i<k\le j}} (Q^m_{ik-1}Q^{qm}_{kj}) && (2.10)\\
&\quad + \sum_{\substack{k\\ i<k\le j}} (c(k-i)Q^{qm}_{kj}) && (2.11)\\
Q^{qm}_{ij} &= Q^{qm}_{ij-1}c(1) && (2.12)\\
&\quad + Q^b_{ij}M(i,j) && (2.13)
\end{aligned}$$

The formulas described above are visualized in Fig. 2.4. $U_1(x)$ represents the score of having x unpaired bases which are not covered in any loop. $U_2(i,j)$ represents the score for having the base pair (i,j) as the outermost base pair. $H(i,j)$, $I(i,j,k,l)$ and $M(i,j)$ represent the score of a hairpin loop from i to j, an interior loop with outer base pair (i,j) and inner base pair (k,l) and a multiple loop under base pair (i,j) respectively. The variables a and c represent the energy of the outer most base pair and the energy of unpaired bases in the case of a multiple loop respectively.
In this work, we will extend the matrices used in McCaskill algorithm and add two new matrices of our own to be able to calculate the probability $P^{\text{loop}}_{(i,j)}(k|S)$ that position k of the sequence is unpaired within a loop closed by the base pair (i,j).

2.3.2 Computation of base pairing probabilities

Base pairing probabilities are computed by a function in the Vienna RNA package. It is mainly predicted from the partition function calculations. The probability that position i and position j are paired together is the summation of the probabilities that they are paired in a Hairpin loop, Interior/Internal loop or are part of a Multiple loop. The probability of the base pair $p = (i,j)$ in a Hairpin loop is the product of the sum of the Boltzmann weights for all structures from 1 to $i-1$ and the sum of the Boltzmann weights for all structures from $j+1$ to n, where n is the length of the sequence, divided by the sum of the Boltzmann weights for all possible structures from 1 to n multiplied by the Hairpin loop energy. For Interior loop, the probability of base pair $p = (i,j)$ is the summation of probabilities of all i‘ and j‘, where i‘ and j‘ are a base pair and lie nested within i and j thus forming an Interior loop, multiplied by the Interior loop energy. Finally for the Multiple loop energy, it is the summation of the probabilities of all nested structures that are part of a Multiple loop multiplied by the energy for the Multiple loop itself.
Another probability calculated by the Vienna RNA package is the stacking probability $sp(i,j)$; which is the probability that base pairs (i,j) and $(i+1, j-1)$ occur simultaneously. This probability is equal to $Q^b_{i+1,j-1}/Q^b_{i,j}$ where Q^b is the McCaskill matrix described in subsection 2.3.1.
Finally, the conditional probability; it is the probability that base pairs (i,j) are stacked i.e. the probability $Pr[(i,j)|(i+1,j-1)]$. This is calculated as the stacking probability of base pair (i,j) divided by the base pair probability of $(i+1, j-1)$.

2.3.3 Chaining Algorithm

Given two RNA sequences S_1 and S_2 and a maximal set of non-overlapping Exact Pattern Matching (EPM)s for these two sequences $E_{1,2}$, we want to find the set X_{EPM} which is a subset of $E_{1,2}$, has maximal size, and contains only non-crossing, non-overlapping EPMs. This is referred to as the Longest Common Subsequence for Exact Pattern Matching (LCS-EPM) problem. Fig. 2.5 gives a better illustration to what is meant by crossing and overlapping EPMs.
The algorithm applies a dynamic programming approach in solving this problem. At the boundaries of matched subsequences, gaps can occur thus forming what is referred to as holes within the subsequence. The LCS-EPM is first solved for small holes, by finding the

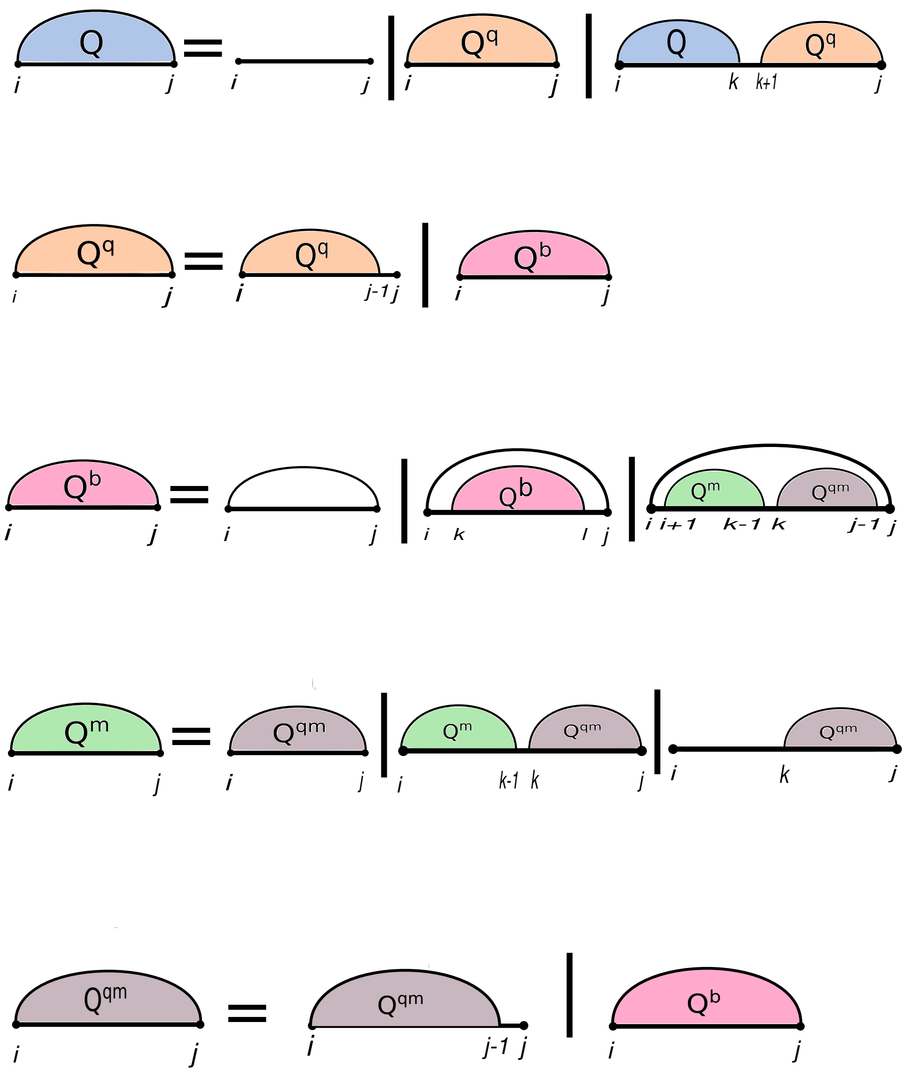

Figure 2.4: An illustration for the recursion used to compute the McCaskill matrices

EPMs that will best fit into these holes. This process is repeated recursively from small holes to bigger ones until the hole with the biggest size is filled. A hole with size equal to the input sequence size is then considered, and the best fitting EPM combination is the set X_{EPM} which will be returned by the algorithm.

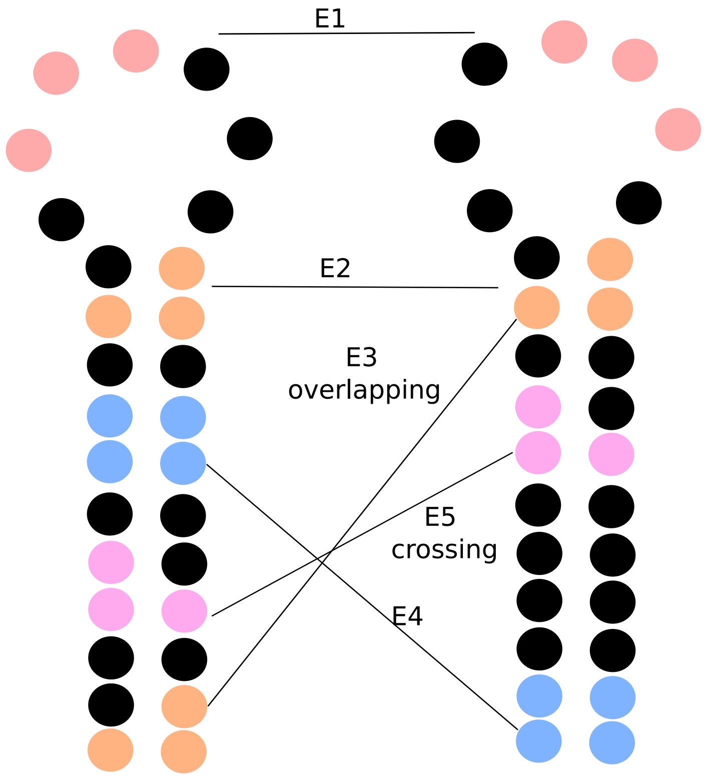

Figure 2.5: This figure shows two RNA structures with the corresponding EPMs. E5 is crossing with E4, so the one which fits best with the other EPMs will be chosen by the algorithm. Also E3 is overlapping with E2, so again only one of them can be picked.

Chapter 3

Exparna-P

This chapter discusses how *ExpaRNA-P* was implemented. It has three main sections, the *Motivation* section which gives an introduction to the algorithmic idea, the *Theory* section will discuss algorithmic details of the implementation. Finally, the *Introduction to Exparna-P* section will give a brief introduction to the tool; stating the input/output formats, extra parameters and how to set their values.

3.1 Theory

3.1.1 Motivation

To solve the LCS-EPM problem, the entire ensemble of structures possible to the given RNA input sequence is considered. Since we need to consider all possible structures, and at the same time we are bound by the time and space complexity of the algorithm. We consider only positions and arcs that have a probability above a certain threshold. Setting such a threshold, reduces the computation time and storage space needed by the algorithm thus producing a faster output.

To start off, two main values are considered, $P\{(i,j)|S\}$ denoting the probability that base pair (i,j) is contained in any structure P of the sequence S. This probability has to be higher than a certain threshold c_p for the base pair to be considered during the calculations. The other value put into consideration is $P^{\text{loop}}_{(i,j)}(k|S)$, which denotes the probability that position k is unpaired within a loop closed by the base pair (i,j) in any structure P of the sequence S. This probability also has to be higher than a certain threshold c_u for that position to be considered during the calculations.
Consider for some (i,j) the list $K = \{k|P^{\text{loop}}_{(i,j)}(k|S) \geq c_u\}$, which is essentially a list containing all unpaired positions within base pair (i,j) with probability higher than c_u. This list is sorted in increasing order. A mapping function $pos^S_{p=(i,j)}(x)$ maps the x-th element of the loop closed by the base pair (i,j) in the list K to its position k in the sequence S. This function aids in minimizing the storage space required by the algorithm, as only relevant positions will be stored and at the same time, their initial sequence position can be looked up in constant time.

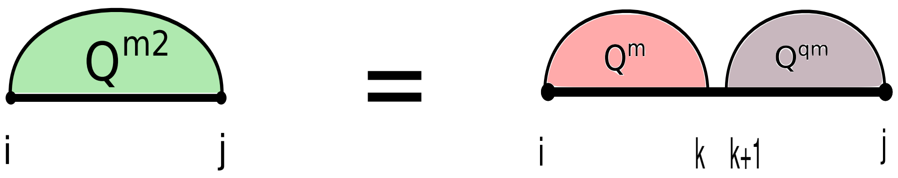

Figure 3.1: The recursion used for calculation of Q^{m2} matrix

3.1.2 Precomputing likely loops

The $P^{\text{loop}}_{(i,j)}(k|S)$ is calculated for all k, for all base pairs with probability higher than c_p. For this, the matrices Q_{ij}, Q^b_{ij}, Q^m_{ij}, and Q^{qm}_{ij} computed by McCaskill's algorithm (see section 2.3.1) are used.
However, the Q^{qm}_{ij} matrix is not stored in the implementation of the Vienna RNA package, so it had to be recalculated. In addition to the McCaskill matrices, the following matrix is computed; $Q^{m2}_{ij} = \sum^{k}_{i<k<j-1} Q^m_{ik} Q^{qm}_{k+1,j}$ representing parts of a multi-loop with at least two outermost base pairs. Fig. 3.1 illustrates the recursion used for $Q^{m2}_{i,j}$ calculation.
Given those matrices, $P^{\text{loop}}_{(i,j)}(k|S)$ can be computed as

$$P^{\text{loop}}_{(i,j)}(k|S) = \left(\frac{H+I+M}{Q^b_{ij}}\right) \cdot P\{(i,j)|S\} \tag{3.1}$$

$$\text{where} \quad H = \exp(-\beta F_1(i,j)) \tag{3.2}$$

$$I = \sum_{\substack{i',j' \\ k<i'<j'<j}} \left(\exp(-\beta F_2(i,j,i',j')) \cdot Q^b_{i'j'}\right) \tag{3.3}$$

$$+ \sum_{\substack{i',j' \\ i<i'<j'<k}} \left(\exp(-\beta F_2(i,j,i',j')) \cdot Q^b_{i'j'}\right) \tag{3.4}$$

$$M = Q^{m2}_{k+1j-1} \exp(-\beta(a+(k-i)c) \tag{3.5}$$

$$+ Q^{m2}_{i+1k-1} \exp(-\beta(a+(j-k)c) \tag{3.6}$$

$$+ Q^m_{i+1k-1} Q^m_{k+1j} \exp(-\beta(a+c)) \tag{3.7}$$

H, I, and M represent the cases where k is contained in a hairpin, interior loop, and multi-loop, respectively. F_1 and F_2 are the energy functions for hairpins and interior loops as specified by McCaskill . The three sums in the computation of M cover the cases where k is in the leftmost (line 3.5), the rightmost (line 3.6), and any other unpaired region (line 3.7) of the loop, respectively. Note that Q^{m2} is required to ensure that the multi-loops considered in M are actually multi-loops, i.e. contain at least two inner base pairs. The formulas are visualized in Fig. 3.2.

3.1.3 Computing the exact pattern

The algorithm first computes for all arc matches, with a probability higher than c_p, the best exact matching for the arcs and the loops enclosed by them and stores the score of

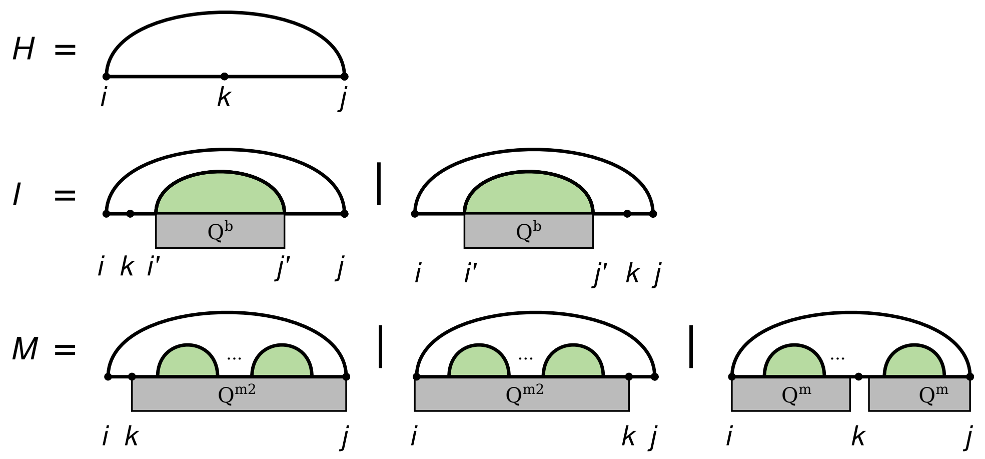

Figure 3.2: Computation of the probability that a position k occurs inside a hairpin loop (H), interior loop (I), or multi-loop (M) closed by a base pair (i,j).

matching the positions within the loop plus the probabilities for the matched base pairs, in a table $R(p_1, p_2)$; where p_1 and p_2 are the arcs in sequence S_1 and S_2 respectively. Three temporary matrices A, G, and B are used to aid in the calculation of the R matrix.The computation starts with the A matrix running from left to right over the two sequences. Then at some point, a gap/jump is allowed using the matrix G, and then start matching the remaining part on the right end of the arc match using the matrix B. Thus an entry $A(x, y)$ denotes the score of the best exact matching of the loops starting from their left ends and ending at position x for sequence S_1 and position y for sequence S_2, respectively. An entry in $G(x, y)$ is the score from the left end of the loop, up to position x for sequence S_1 and position y for sequence S_2 or any smaller position, thus allowing a gap in the matching. $B(x, y)$ is basically the same as $A(x, y)$ except that the matching up to positions x and y may not be connected i.e. can contain a gap somewhere due to the presence of the gap matrix G.

The matrices are computed as specified in Fig. 3.3, the base cases are $A(0, 0) = 0$, $A(0, x) = A(x, 0) = -\infty$ for all $x > 0$, $B(0, x) = B(x, 0) = G(0, x) = G(x, 0) = 0$ for all x. For the A matrix, the recursion assigns $-\infty$ to positions x and y if they do not match (first case), or assigns the score of the last matching position $A(x‘, y‘)$ plus the score of matching this one in case they are adjacent (i.e. $x‘ = x - 1$ and $y‘ = y - 1$), which represents the second case. Or, the last case which represents a match of the two base pairs. For the last case, we loop over all arcs with right end x for S_1 and y for S_2. Let $pos^{S_1}_{p_1}(x') = k_1$, $pos^{S_2}_{p_2}(y') = k_2$, $pos^{S_1}_{p_1}(x) = l_1$ and $pos^{S_2}_{p_2}(y) = l_2$. If $k_1 - 1$ is the right end of the arc for p_1, $k_2 - 1$ is the right end of the arc for p_2, $l_1 + 1$ is the left end of the arc for p_1 and $l_2 + 1$ is the left end of the arc for p_2, then the score for $A(x, y)$ is set to the score of matching from the left end of the arc $A(x‘, y‘)$ plus the score of matching underneath the arc $R(p‘_1, p‘_2)$ plus the score of matching the right end of the arcs using the $match(x, y)$ function; which is the score of matching the sequence positions plus the base pairing probabilities for both arcs plus the stacking probabilities. The recursion of the B matrix is the same except that it considers the case that a gap has been added at the end of positions x and y. For the G matrix, we consider the best score of either adding no gaps, or adding a gap in only one sequence or in both.

After the R matrix is filled for all arc matches, two final matrices which are relatively similar to each other, F matrix and $Trace$ matrix, are computed. An entry in the F matrix, $F(i, j)$, has the best matching score ending at position (i, j) where i is in S_1 and j is in S_2. The $Trace$ matrix is the reverse of the F matrix; an entry $Trace(i, j)$ denotes the best matching score from positions (i, j) to the end of both sequences. For each position in the $Trace$ matrix, there is a pointer to the next matching position which would maximize the matching score. The pointer is assigned to $-\infty$ if there is no possible matching from the current position. The $Trace$ matrix is computed for tracing back the calculated EPMs, as it makes the implementation of the Chaining algorithm easier since an EPM score would be the score from the start position of the EPM, till the end of the sequence.
The recursion used for the computation of the F matrix is shown in Fig. 3.4. An entry $F(i, j)$ is given a score 0 if $S_1[i] \neq S_2[j]$ (first case), or the score of the previous entry plus 1, if $S_1[i] = S_2[j]$ which represents the case of sequential matching (second case). The last case represents structural matching, where i and j are the right ends of the arc, so the score of $F(i, j)$ is set to the score of the arc match plus matching the unpaired bases underneath the arc match plus the score of matching from the beginning to the left end of the arc. Relational weights are used to favor certain types of matchings over others. These are represented by α_1 for sequential score, α_2 for structural score and α_3 for stacking. This is done for all arcs which have a right end (i, j) and the best score is the one used.

Now that all the matrices are filled, we loop over the $Trace$ matrix and if the current position has a score higher than or equal to a certain threshold for EPMs that position is stored in a list L. By the end of this loop, the list L would contain the start positions of all possible EPMs. The list is sorted according to the score in descending order. For every position in this list, an EPM is constructed if possible by tracing the next matching position from it. Each position can have one of three values; $-\infty$ means that this position is the end of the matching, ∞ means that this position was used in a previous EPM i.e. this EPM overlaps with another EPM. And finally any other integer value which means that this position is not included in any EPM. For each position in the list L, we check first if the score of the current element is not $-\infty$. If it is then this position is disregarded and the next one from the list L is picked. If it is not, then we loop over the $Trace$ matrix, every time checking if there exists a pointer from the current position to a next one. If a pointer does not exist then we break from the loop and begin the process for the next element in the list. However if such a pointer exists, then we check whether there exists a sequential or a structural matching from this position and update the EPM structure accordingly. The score of the current position is set to ∞ and the pointer of the current element is updated to point to the element next to the current one. If a position with score ∞ is encountered during the loop, the entire EPM is disregarded and the scores of the positions used in it are reset to their initial values. Every EPM computed is then added to a list which is later on passed to the Chaining algorithm.

$$A(x,y) = \max \begin{cases} -\infty \\ \\ \text{if adjacent}(x,y) \text{ then} \\ \qquad A(x-1,y-1) + \text{match}(x,y) \\ \\ \text{for all } p'_1 = (x',x) \in P_1, p'_2 = (y',y) \in P_2, x', y' \\ \;\text{with } \text{pos}^{S_1}_{p_1}(x) - 1 = {p'_1}^R \text{ and } \text{pos}^{S_1}_{p_2}(y) - 1 = {p'_1}^R \\ \;\text{and } \text{pos}^{S_1}_{p_1}(x') + 1 = {p'_1}^L \text{ and } \text{pos}^{S_2}_{p_2}(y') + 1 = {p'_2}^L \\ \qquad A(x',y') + R(p'_1,p'_2) + \text{match}(x,y) \end{cases}$$

$$G(x,y) = \max\{A(x,y), G(x-1,y-1), G(x-1,y), G(x,y-1)\}$$

$$B(x,y) = \max \begin{cases} G(x,y) \\ \\ \text{if adjacent}(x,y) \text{ then} \\ \qquad B(x-1,y-1) + \text{match}(x,y) \\ \\ \text{for all } p'_1 = (x',x) \in P_1, p'_2 = (y',y) \in P_2, x', y' \\ \;\text{with } \text{pos}^{S_1}_{p_1}(x) - 1 = {p'_1}^R \text{ and } \text{pos}^{S_1}_{p_2}(y) - 1 = {p'_1}^R \\ \;\text{and } \text{pos}^{S_1}_{p_1}(x') + 1 = {p'_1}^L \text{ and } \text{pos}^{S_2}_{p_2}(y') + 1 = {p'_2}^L \\ \qquad B(x',y') + R(p'_1,p'_2) + \text{match}(x,y) \end{cases}$$

where

$$\text{match}(x,y) := \begin{cases} 2 \cdot \alpha_1 + (P\{(x',x)|S_1\} + P\{(y',y)|S_2\}) \cdot \alpha_2 + (sp(x',x) + sp(y',y)) \cdot \alpha_3 \\ \;\text{if } S_1(\text{pos}^{S_1}_{p_1}(x)) = S_2(\text{pos}^{S_2}_{p_2}(y)) \\ -\infty \text{ otherwise} \end{cases}$$

$$\text{adjacent}(x,y) := \quad (\text{pos}^{S_1}_{p_1}(x) - 1 = \text{pos}^{S_1}_{p_1}(x-1) \text{ and } \text{pos}^{S_2}_{p_2}(y) - 1 = \text{pos}^{S_2}_{p_2}(y-1))$$

Figure 3.3: Recursions for matrices A, G, and B

$$F(i,j) = \max \begin{cases} 0 \text{ if } S_1[i] \neq S_2[j] \\ \\ \text{if } S_1[i] = S_2[j] \text{ then} \\ \qquad F(i-1, j-1) + 1 \\ \\ \text{for each } p_1 = (i`, i) \in P_1, p_2 = (j', j) \in P_2 \\ \qquad F(i`-1, j`-1) + \text{amScore}(p_1, p_2) \end{cases}$$

where

$$\text{amScore}(p_1,p_2) := \quad R(p_1,p_2) + 2 + P\{(i`,i)|S_1\} + P\{(j`,j)|S_2\}) \cdot \alpha_2 + (sp(x',x) + sp(y',y)) \cdot \alpha_3$$

Figure 3.4: Recursions for F matrix

3.1.4 Chaining algorithm

By the end of the calculations described in the previous subsection, we have a list which contains a set of non-overlapping, maximally extended EPMs along with their scores and structures. This list is passed as input to the chaining algorithm and as output we get a maximum combination of non-crossing, non-overlapping EPMs (section 2.3.3). A graphical output is also produced by the folding algorithm in the Vienna RNA Package. The input sequences are folded into their most probable secondary structure, putting into consideration that the pattern matches produced by the chaining algorithm should be kept as is i.e. in the structural matches, the base pairs forming the arc match should be paired by the folding algorithm, and in case of sequential matching, the unpaired positions should remain unpaired. Fig. 3.5 illustrates a graphical output from the algorithm. As we can see in the figure, in the case of the EPM colored in red that gaps can exist within the matching. EPMs must be connected either on a sequential or a structural level, and since the positions cannot be sequentially matched due to the gap in the middle. This means that the positions are matched structurally to each other. However the base pair connecting the two parts has low probability, and hence is not considered by the folding algorithm resulting in the gap in the middle of the EPM.

3.2 Introduction to Exparna-P

3.2.1 Installation Requirements

To install *ExpaRNA-P*, the Vienna RNA package should be installed, which can be found at the following url: `http://www.tbi.univie.ac.at/~ronny/RNA/`. The Vienna RNA package requires the G2 library to be installed first. The G2 library can be found under the following url: `http://sourceforge.net/projects/g2/`. The Vienna RNA package should be configured with the following options: *–without-forester –without-perl –disable-openmp*. For the installation of *ExpaRNA-P*, the option *–with-vrna* needs to be specified followed by the path of where the Vienna RNA package was installed.

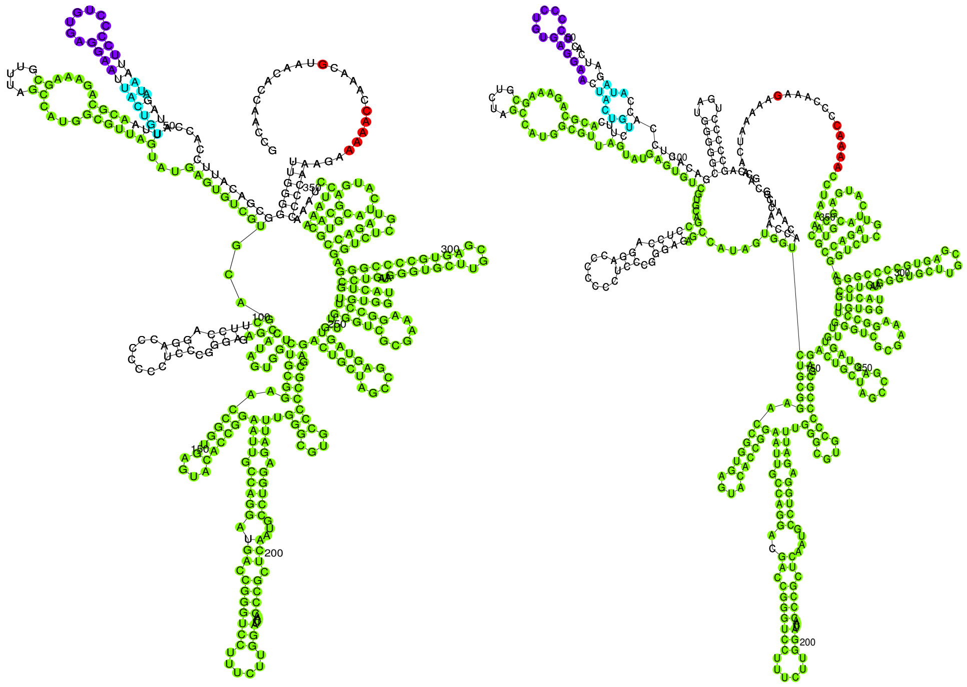

Figure 3.5: The figure shows the output from running the *ExpaRNA* tool on two RNA sequences with GenBank codes: AF165050 (bases 1 – 379) and D45172 (bases 1 – 391), with all the parameters set to their default values. The structure on the left comes from the RNA with GenBank code: AF165050, and the one on the right: D45172. Matching EPMs are given in the same color.

The tool takes as input two files in Clustal format, each containing the RNA sequence. The output is two post script files for the two sequences, containing the most likely structure for the sequence taking into account the EPMs calculated. In each file, matching patterns are given the same colors for ease of identification, and different EPMs are assigned different colors. There is a number of options that can be set which control the output and the run time of the program.

3.2.2 Output file parameters

- There are two parameters for specifying the name of the two post script output files. The default names are "SequenceA" and "SequenceB". This can be changed by specifying *-a* and *-b* options followed by the name desired for the output files for sequences S_1 and S_2 respectively.
- The output from the tool can be used as anchor constraints to be fed into *LocARNA* for better alignment of the two sequences. This can be specified by the command *-i*, a file with the name *locarna_constraints_input.txt* will be produced which contains the anchor constraints in fasta format to be fed into *LocARNA*.

3.2.3 Threshold and probability related parameters

- The minimum probability threshold for a base pair to be considered can be altered by the parameter *-P* followed by the new value. Decreasing the probability increases the run time of the algorithm, as more base pairs will need to be considered during the calculations to find the longest non-overlapping, non-crossing set of EPMs.
- To set the minimum probability threshold for $P^{\text{loop}}_{(i,j)}(k|S)$, use the option *-p* followed by the minimum probability. Increasing the value assigned to the probability, decreases the run time of the algorithm. This is because more positions with probabilities less than the threshold will be eliminated and thus resulting in less computation time and less storage space at run time.
- To specify the maximum allowed distance between (i_1, j_1) and (i_2, j_2), i.e. maximum difference for alignment traces(Fig. 3.6 better illustrates the parameter), set the option *-d* followed by the new distance. Restricting the distance between matches helps in improving the quality of the output. In the case that no restrictions are set to the allowed distance between the alignment traces, an arc at the beginning of one sequence could be matched to an arc at the end of the other sequence. The positions to the right of the right end of the arc in the first sequence and the positions to the left of the left end of the arc in the second sequence have a limited number of possible matchings, as they are restricted by the arc match.
- To specify the maximum difference in the sizes of the arc matches, set the option *-D* followed by the new value (See Fig. 3.7). The sizes of the arcs in an arc match are better kept as close as possible. Consider the case where no restrictions are put for the size difference. Then a big arc in one sequence would be matched to a small one in another. The bases underneath the big arc match cannot be included in

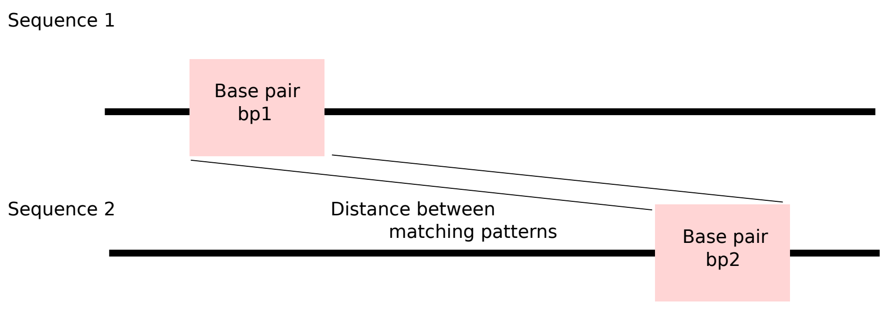

Figure 3.6: The figure above shows two arc matches, the maximum allowed distance restricts the difference between the start position of $bp1$ and the start position of $bp2$, and the end position of $bp1$ and the end position of $bp2$.

any EPM because they would be crossing with the EPM where the arc is matched which would affect the size of the resulting output from the tool.

- To turn the option for calculating and including the stacking probabilities on and off, use the -S option (section 2.3.2).
- To set the minimum score for an EPM to be considered, use the option -t followed by the minimum score.
- To set the minimum size allowed for an EPM, use the option -s followed by the minimum size.

3.2.4 Scoring related parameters

The tool has three parameters related to the scoring of matchings:

- α_1 for sequential score.
- α_2 for structural score.
- α_3 for stacking score.

and according to the values set for them, certain matchings will be favored to others. For instance if α_2 was given value 2 while α_1 and α_3 have values 1, then structural matching will be favored over sequential matching and over stacking whenever possible.

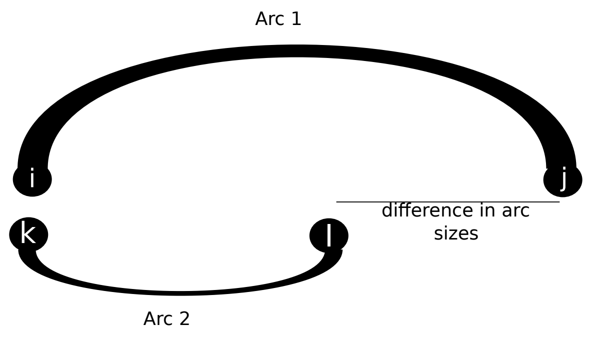

Figure 3.7: The figure above shows two arcs, *Arc 1* and *Arc 2*. The difference in the size of the arcs is measured as the size of the first arc minus the size of the second arc, or in the example above: $(j - i) - (l - k)$

Chapter 4

Implementation

In this chapter, we will discuss the implementation details of the tool. In the Working Environment section, a brief introduction to the programming language the tool was implemented in and the repositories used is provided (section 4.1). Then, the Heuristics section the default values of parameters are stated and the theory behind setting them to those values is explained (section 4.2).

4.1 Working Environment

4.1.1 Programming Language Used

The tool was implemented entirely in C++. C++ was developed in 1979 at Bell Labs by Bjarne Stroustrup as an enhancement to C. It is a general purpose programming language, it supports data abstraction, object-oriented programming and generic programming. It is statically typed, compiled and a multi-paradigm programming language. It is considered as a hybrid/intermediate-level language; as it combines features of both high-level and low-level programming languages [10].

4.1.2 Repositories Used

The code was kept on the Mercurial repository for easier update and integration. Mercurial is a fast, platform independent, extensible, open source, distributed revision control tool for software developers. It is mainly a command line program, where all the operations are invoked by the keyword *hg*, a reference to the chemical symbol of the element mercury. Graphical user interface extensions are also available for use. It is mainly implemented using Python programming language [9].

4.2 Heuristics

In this section, the default values used for the parameters in the tool are stated, providing an explanation as to why these values were chosen.

4.2.1 Threshold and probability related parameters

- The minimum probability threshold *-P* for a base pair to be considered in the calculations, has a default value 0.0074. It was observed during the evaluation of the tool that the higher the probability is set, and when combining the output of the tool with other alignment tools that a more accurate alignment is produced. This could be explained by the fact that with lower probabilities, base pairs which are not very probable or stable in nature are considered in the calculation and thus producing a structure which is not close to the actual one.

- The minimum probability threshold for $P^{\text{loop}}_{(i,j)}(k|S)$, *-p*, has a default value of 0.01. During the evaluation of the tool, it was observed that when increasing the value for the threshold, more accurate alignment results are produced. This can be attributed to the fact that with lower probabilities, more positions are considered as unpaired in the loop which is not preferred in nature, thus the structure produced is less accurate than when increasing the probability threshold.

- The maximum difference for alignment traces *-d*, has a default value -1. This means that there is no maximum difference used, i.e. base pairs can be as far away from each other as possible.

- The maximum difference in the sizes of the arc matches *-D* also has a default value -1. This also means that the difference in the sizes can be as big as possible (no restrictions on the size difference).

- The default value for the flag of computing the stacking probabilities *-S* is off. Stacking probabilities are not computed by default and not used.

- The default value for the EPM score threshold *-t* is 5. This is because we only want to consider structures that consist of at least 5 unpaired nucleotides in case of sequential matching or with structural matching of non-nested base pairs. Lowering the value results in more matching, however it slows down the tool and it is not very accurate since EPMs of mainly unpaired bases of sizes less than 5 would be produced.

- The default minimum size *-s* for an EPM is 3. If the default size of the EPM is not changed, then the EPM score threshold is used to eliminate EPMs with lower score. However, if the minimum size gets a value different from 3, then we consider only the EPM size when eliminating EPMs.

4.2.2 Scoring related parameters

For the three scoring parameters α_1, α_2 and α_3, the default value is set as 1. Thus no type of matching is favored over the other types.

Chapter 5

Evaluation

This chapter describes the evaluation procedure used to analyze the results produced from the implemented tool. We used the same evaluation procedures to evaluate our tool, as the ones used for *ExpaRNA*. The evaluation is divided into two sections. The first section handles the small scale evaluation where we inspect the output of the tool for medium/large sized RNA sequences. In the second section, we combine *ExpaRNA-P* with *LocARNA* and compare the results of the alignment produced with the results of combining *ExpaRNA* with *LocARNA* using the compalign score which will be further explained below.

5.1 Small scale evaluation

The same evaluation technique is used as the one used for *ExpaRNA*. We have chosen two pairs of RNAs: (a) two IRES RNAs from hepatitis C virus, which belong both to the Rfam family HCV_IRES for IRESs [3]. GenBank: AF165050 (bases 1 – 379) and D45172 (bases 1 – 391). (b) Two 16S rRNAs. The first RNA is from *Escherichia coli* and is 1541 bases long. The second RNA of length 1551 stems from *Dictyostelium discoideum* (GenBank codes: J01859 and D16466). For the comparison, we used the EPM size 2 which is the same as the one used in the *ExpaRNA* evaluation. The output from the tool for the two IRES RNA sequences is shown in Fig. 5.1, and for the 16S rRNAs is shown in Fig. 5.2. As we can see in Fig. 5.1, within an EPM a gap can exist (section 3.1.4).
Table. 5.1 shows a comparison of the output results produced by our tool, *ExpaRNA*, *RNA_align* and *RNAforester* [13], [5]. The number of matches and percentage coverage for *ExpaRNA*, *RNA_align* and *RNAforester* were obtained from the paper by Heyne et.al. [13]. The *RNA_align* uses the general edit distance algorithm by Jiang et.al. [5] to compute the sequence-structure alignment for RNA sequences. On the other hand, the *RNAforester* program extends the tree editing algorithm of Jiang et.al. to calculate forest alignments [4], [7]. The comparison is based on the total number of matched nucleotides by each method, and by the percentage coverage. The coverage rate is twice the number of EPMs produced divided by the sum of the two sequence lengths. As we can see from the table, the coverage rate for *ExpaRNA-P* is better than the other tools for the *IRES RNA* sequences. On the other hand, with the *16S rRNA* sequence, the coverage of *ExpaRNA-P* is worse than all tools. Table. 5.2 shows a comparison between the run times of *ExpaRNA* and *ExpaRNA-P* for both RNA families. The run time of

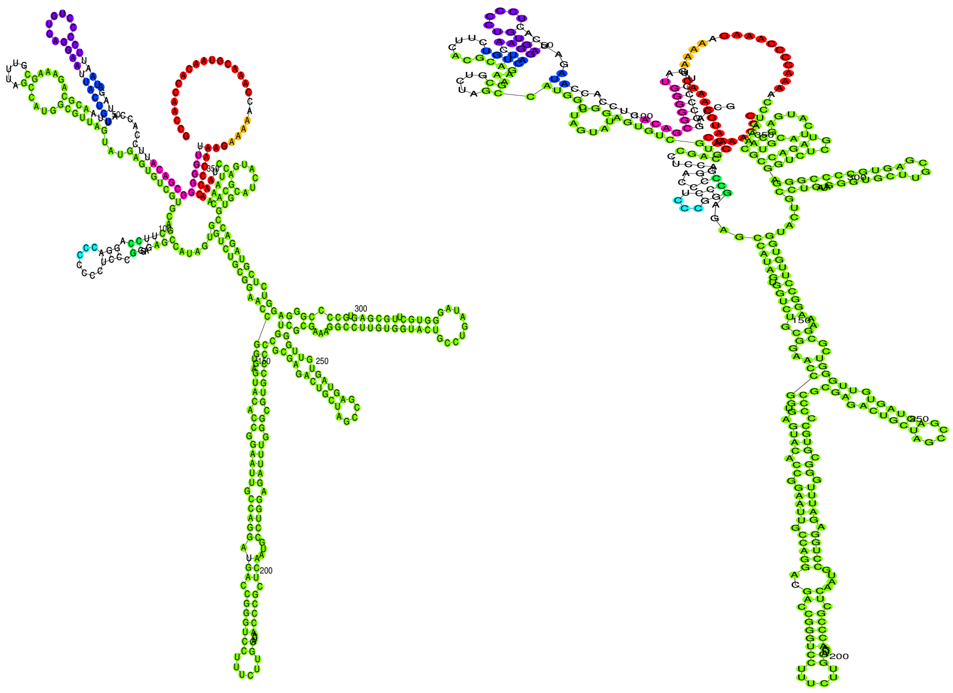

Figure 5.1: The output produced for the two IRES RNA sequences. On the left we have the structure of the RNA with GenBank code: AF165050, and on the right: D45172. Matching EPMs are given the same color in both structures, and each EPM is shown in a different color. We have a coverage of 86.5% (333 matched bases)

ExpaRNA-P appears to be worse than that of *ExpaRNA*, but that is expected since it considers all the possible structures for an RNA sequence, not a single fixed structure like *ExpaRNA*.
Nonetheless, this evaluation cannot be considered as accurate, since it is not known if a bigger coverage is better or not for the sequences. In addition to that, there is no reference to which the structures produced by the tool can be compared against. The above reasons provided motivation for performing the large scale evaluation which is described in the next section.

Method	IRES RNA		16S rRNA	
	No of Matches	Coverage(%)	No of Matches	Coverage(%)
ExpaRNA-P	333	86.5	771	49.87
ExpaRNA	175	45	875	57
RNA_align	192	50	861	56
RNAforester	128	33	847	55

Table 5.1: Comparison between the number of matchings and the percentage coverage found by *ExpaRNA-P* and other alignment methods

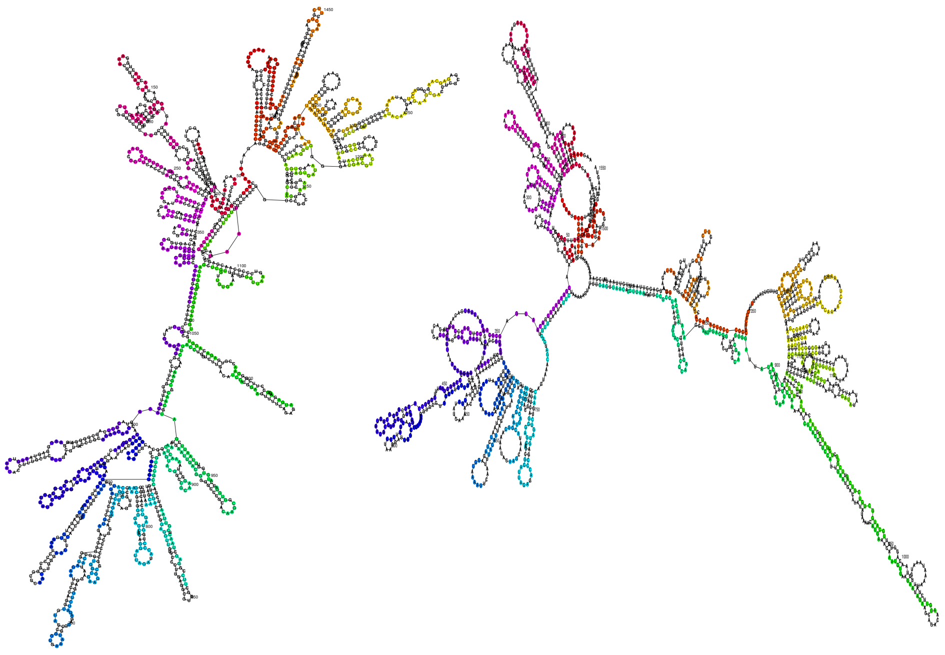

Figure 5.2: The output produced for the two 16S rRNA sequences. On the left we have the structure of the RNA with GenBank code: J01859, and on the right: D16466. Matching EPMs are given the same color in both structures, and each EPM is shown in a different color. We have a coverage of 49.87% (771 matched bases)

Method	IRES RNA		16S rRNA	
	No of Matches	Time	No of Matches	Time
ExpaRNA-P	333	2.91 s	771	2 m 07 s
ExpaRNA	175	0.67 s	875	4.66 s

Table 5.2: Comparison between the run time of *ExpaRNA-P* and *ExpaRNA*

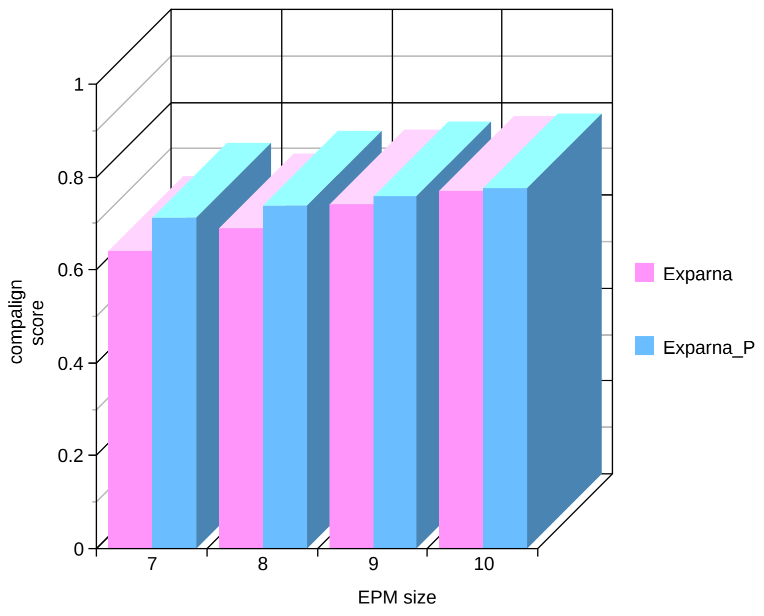

Figure 5.3: A graph visualizing the comparison between *ExpaRNA* and *ExpaRNA-P* for all sequences of the k2 dataset. The x-axis shows the minimum EPM size. The y-axis shows the average compalign score.

5.2 Large scale evaluation

The output from the tool can be used as anchor constraints for RNA alignment tools like *LocARNA*, thus speeding up the overall performance of such tools. In this section, we compare the performance of *ExpaRNA-P* combined with *LocARNA* in comparison to *ExpaRNA* combined with *LocARNA*. The input set to both tools was the k2 dataset from the Bralibase 2.1 benchmark [2], [6]. For each input file, *ExpaRNA-P* is run, and the resulting EPMs are provided as anchor constraints to *LocARNA* to produce the resulting alignment. We compare the resulting test alignment with the reference alignment, which is available with every sequence in the k2 dataset of the Bralibase 2.1 benchmark, using the Compalign score. The Compalign score measures how close the test alignment is to the reference alignment [1], [2], [6].
This procedure was repeated for each input file four times on the following EPM minimum sizes $s = (7, 8, 9, 10)$. Figure 5.3 summarizes the results for the entire dataset.

We observe from Fig. 5.3 that the alignment produced using the EPMs produced by *ExpaRNA-P* as anchor constraints to *LocARNA* is more accurate than the alignment produced using the EPMs from *ExpaRNA* as anchor constraints to *LocARNA*. However we can also notice that with increasing the minimum EPM size, the compalign score produced by *ExpaRNA* increases at a more rapid rate than the compalign score produced by *ExpaRNA-P*. In addition to that, we can see from Fig. 5.4 and Fig. 5.5, the accuracy of the score produced depends mainly on the RNA family; for the *HIV_PBS* family, the compalign score produced by *ExpaRNA-P* is almost close to 1. On the other hand, for the *Cobalamin* family, the compalign score is below 0.5 for both tools.

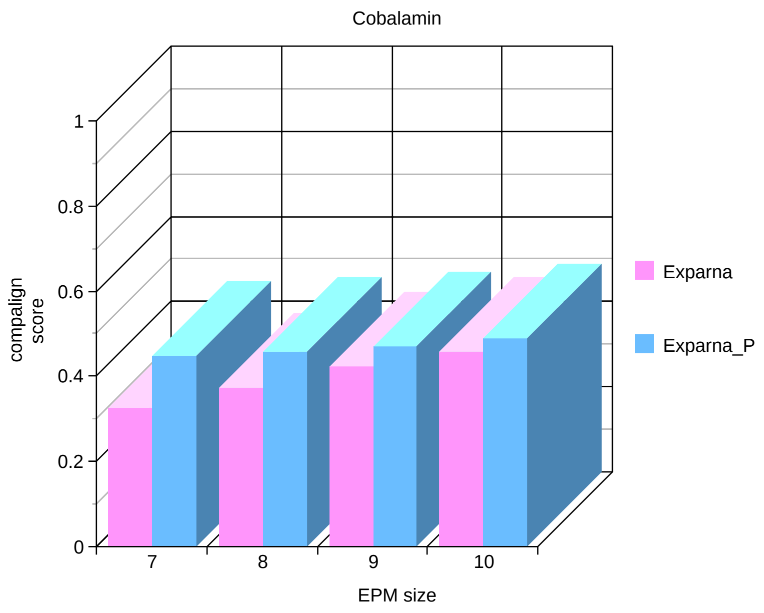

Figure 5.4: A graph visualizing the comparison between *ExpaRNA* and *ExpaRNA-P* for the RNA family *Cobalamin*. The x-axis shows the minimum EPM size. The y-axis shows the average compalign score.

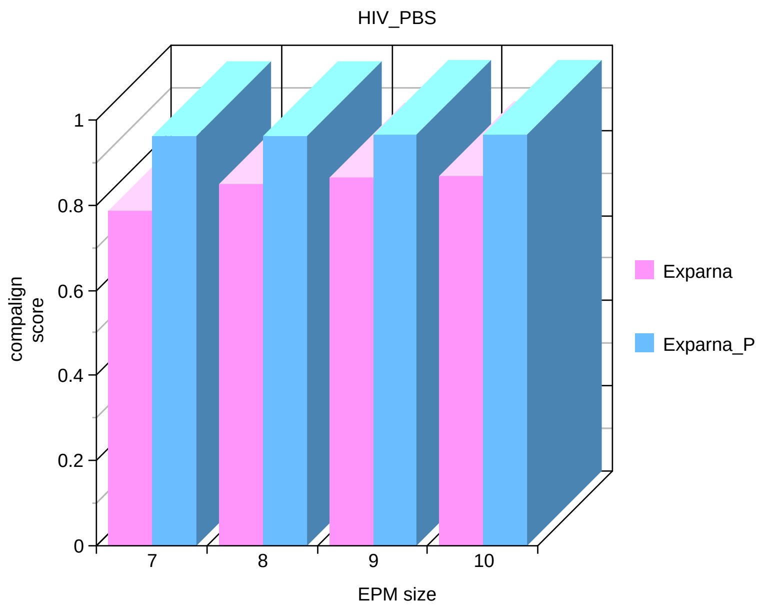

Figure 5.5: A graph visualizing the comparison between *ExpaRNA* and *ExpaRNA-P* for the RNA family *HIV_PBS*. The x-axis shows the minimum EPM size. The y-axis shows the average compalign score.

Chapter 6

Conclusion

6.1 Summary

We created a tool that performs sequence-structure comparison for RNA sequences. This tool, given two RNA sequences outputs the maximal non-crossing, non-overlapping set of exact pattern matchings. We used a dynamic programming approach to calculate the exact pattern matches and to produce the set with maximal cardinality. The output from the tool could be used as constraints for RNA alignment tools, restricting their search space and thus speeding them up.
To evaluate the outcome produced, we compared the output of the tool with other tools like *ExpaRNA* which also perform sequence-structure comparison but use different algorithms in their computation. The comparison was based on the coverage produced by each tool for the sequences, and throughout this comparison we could see that the results cannot be considered as conclusive since for one input *ExpaRNA-P* had higher coverage than the other tools, while for the other it had lower coverage. This and other factors triggered the need for performing a more accurate and large scale evaluation. When combining *ExpaRNA-P* with *LocARNA* and comparing it with *ExpaRNA* combined with *LocARNA* against the Bralibase benchmark, the average compalign score produced by *ExpaRNA-P* was more accurate than *ExpaRNA*. In spite of the more accurate alignment results by *ExpaRNA-P*, we observed that when increasing the minimum EPM size, the rate of increase of the compalign score produced by the results of *ExpaRNA*, is higher than the rate of increase of accuracy from the results of *ExpaRNA-P*.

6.2 Future Work

As a result of time insufficiency, comparison based on the run time during the benchmarking phase was not carried out. Work needs to be done to compare the run times of both *ExpaRNA* and *ExpaRNA-P*. We also need to identify the possible speedup produced from altering the minimum EPM size, and also compare it with the results from *ExpaRNA*. More work is to be done to identify whether the current default values to the parameters are the best compromise between speedup and accuracy, or other values produce better results. We need to explore reasons why the rate of increase of the compalign score produced by *ExpaRNA* is higher than that of the results of *ExpaRNA-P*, when increasing the minimum EPM size.

Appendix A

Abbreviations

RNA	RiboNucleic Acid
A	Adenine
U	Uracil
C	Cytosine
G	Guanine
EPM	Exact Pattern Matching
MFE	Minimum Free Energy
LCS-EPM	Longest Common Subsequence for Exact Pattern Matching

Bibliography

[1] Bahr A. et al. Balibase (benchmark alignment database): enhancements for repeats, transmembrane sequences and circular permutations. *Nucleic Acids Res.*, 29:323–326, 2001.

[2] P.P.Gardner et al. A benchmark of multiple sequence alignment programs upon structural RNAs. *Nucleic Acids Res.*, 33:2433–2439, 2005.

[3] S. Griffiths-Jones et al. Rfam: annotating non-coding RNAs in complete genomes. *Nucleic Acids Research*, 33, 2004.

[4] T. Jiang et al. Alignment of trees - an alternative to tree edit. *Theoritical Computer Science*, 143:137–148, 1995.

[5] T. Jiang et al. A general edit distance between RNA structures. *Computaional Biology*, 9:371–388, 2002.

[6] Wilm A et al. An enhanced RNA alignment benchmark for sequence alignment programs. *Algorithms Mol. Biol.*, 1, 2006.

[7] Giegerich R Kurtz S Hochsmann M, Toller T. Local similarity in RNA secondary structures. *Proceedings of the IEEE Bioinformatics Conference*, pages 159–168, 2003.

[8] McCaskill J.S. The equilibrium partition function and base pair probabilities for RNA secondary structure. *Biopolymers*, 29:1105–1119, 1990.

[9] Matt Mackall. Mercurial v0.1 - a minimal scalable distributed SCM. April 20, 2005.

[10] Herbert Schildt. *C++ The Complete Reference Third Edition.* Osborne McGraw-Hill, August 01, 1998.

[11] Ivo L. Hofacker Peter F. Stadler Sebastian Will, Kristin Reiche and Rolf Backofen. Inferring non-coding RNA families and classes by means of genome-scale structure-based clustering. *PLOS Computational Biology*, 3(4), 2007.

[12] Butcher SE Staple DW. Pseudoknots: RNA structures with diverse functions. *PLOS Biology*, 3(6), June, 2005.

[13] Michael Beckstette Rolf Backofen Steffen Heyne, Sebastian Will. Lightweight comparison of RNAs based on exact sequence-structure matches. *Bioinformatics*, 25(16):2095–2102, February 2, 2009.

[14] Hofacker I Schuster P Wuchty S, Fontana W. Complete suboptimal folding of RNA and the stability of secondary structures. *Biopolymers*, 49:145–165, 1998.

[15] Michael Yarus. Getting past the RNA world: The initial darwinian ancestor. *Cold Spring Harbor perspectives in biology*, April 21, 2010.