**Media Engineering and Technology Faculty**
**German University in Cairo**

# 3D-Structure-Motifs Aware Sequence Structure Alignment of RNAs

**Bachelor Thesis**

| | |
|---|---|
| Author: | Mounir Stino |
| Supervisors: | Prof. Dr. Rolf Backofen |
| | Dr. Sebastian Will |
| Reviewer: | Prof. Dr. Slim Abdennadher |
| Submission Date: | 29 August, 2007 |

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

Mounir Stino
29 August, 2007

# Acknowledgments

Many people helped to make this project possible. First I wish to thank Prof. Dr. Slim Abdennadher, the dean of the Faculty of Media Engineering and Technology and the head of the Computer Science department in the German University in Cairo, who despite his responsibilities has managed to provide helpful guidance in choosing my project. He has also guided me with patience throughout the years of my bachelor study.

Many thanks go to my supervisors, Prof. Dr. Rolf Backofen and Dr. Sebastian Will, for introducing me to the field of bioinformatics and for helping me throughout the project. Thanks to Monika Degen-Hellmuth for accomplishing all the administrative work. I would also like to thank the German Academic Exchange Service (DAAD) and the German University in Cairo for providing me a scholarship during the six months I stayed in Germany.

In addition I would like to thank Fabrice Jossinet for providing us the data needed to accomplish this project and for his hospitality during our short visit to Strasbourg.

Special acknowledgments to all members of the department of bioinformatics at the University of Freiburg and to my roommates who I consider to be my close friends and extended family in Germany.

Finally and most importantly thank you to the greatest parents; I wouldn't be able to go through every moment of happiness and hard work in education and life without your support and love.

# Abstract

Comparison of RNAs is mainly based on information about the sequences and their secondary structure. The function of the RNAs on the other hand is based on their 3D-structure, which is hard to determine. However, there are wide-spread 3D-motifs which can be identified more easily. Such motifs can be defined as an ordered assembly of non-Watson-Crick base pairs. Current sequence structure alignment methods are not aware of such motifs; however, these motifs can give strong guidance for such alignments. The detection of the motifs is done in several steps. First, a list of isosteric motifs, which are motifs given in sequence level that have the same 3D structure, is produced. Then, string searching algorithms are implemented to search for the motifs in the RNA sequences. Finally, the motif search is integrated in *LocARNA*, an already existing sequence-structure alignment tool. The modified alignment algorithm includes matching structural motifs.

# Contents

# Chapter 1

# Introduction

Research in molecular biology is generating enormous amounts of data that are not manageable by hand; hence the new field of bioinformatics has emerged. The field of bioinformatics, also known as computational biology, is about using computer systems to analyze and extract valuable information from massive amounts of biological data. An example of the use of computers in the analysis procedure is the design of efficient algorithms to investigate large DNA, RNA and protein sequences.

The recent discoveries of the roles that RNA plays within cells made it the focus of a lot of research after it was underestimated for a long time. An RNA molecule is a sequence of bases of four possible types, denoted by the letters A, C, G and U, connected by a backbone. The primary structure describes an RNA on the sequence level, as a sequence of bases. The secondary structure of the RNA is a list of bonds formed between the bases of the molecule. Methods for determining the secondary structure of an RNA molecule in a biological lab are expensive. However, efficient algorithms mostly based on dynamic programming were developed to predict the secondary structure of RNA[7]. During the evolution of RNA molecules, a lot of mutations to the bases occur, such as insertions, substitutions and deletions, changing the primary structure. The secondary and tertiary structures however are conserved.

The function of the RNA within the cell is determined in large part by the 3D structure of the RNA molecule when it folds. Determining the tertiary structure of an RNA molecule in a biological laboratory is hard and time consuming. Only a small subset of RNAs has a known tertiary structure. The problem of predicting the 3D structure is NP complete, making it practically impossible for large RNA molecules[14].

The problem of comparing RNAs, known as RNA alignment, is a well known and studied problem in bioinformatics. Aligning a molecule with one of known function can give a guidance of its function within the cells. It can also determine evolutionary facts, and arrange the RNAs to families. Alignment algorithms present today match the sequence and the structure of RNAs. This means that the alignment is done on the primary and secondary structure levels, which do not completely reflect the function of the molecule. While observing the known 3D structures of some crystallized RNA molecules, highly conserved structural motifs were noticed. These motifs have complex architectures in which a large fraction of the bases engage in non-Watson–Crick base pairs. The motifs' architecture is highly conserved during evolution. The motifs are operationally defined as "ordered arrays of non-Watson–Crick base pairs[6]." Non-Watson–Crick base pairs are divided to twelve distinct geometric families according to the orientation of the glycosidic bonds of the interacting bases. Within each family isosteric subgroups are observed. All the base pairs in an isosteric subgroup have roughly the same molecular distance, and thus the bases can be substituted without affecting the 3D structure.

A sequence signature of a motif is the set of nucleotide sequences that fold to form the same 3D motif. The set can be generated combinatorially knowing the type of bonds within the bases of the motif. Being described on the sequence level, the detection and matching of the sequence signature of a motif can be integrated in sequence alignment software.

The main task of this project is to integrate 3D motifs detection and matching into LocARNA[14], an already existing sequence-structure alignment tool. This is done in several steps. First, a list of isosteric motifs is generated knowing the type of bonds between the bases of the motif and the isostericity matrices of each family of non-Watson–

Crick base pairs. Next is to search for these motifs in the RNA sequences, using an efficient string searching algorithm. The final step is to match the found motifs during the alignment of the sequence and structure of the RNAs.

The rest of the document will be organized as follows: Next chapter presents the biological background needed to understand the work. Then, a brief explanation of *LocARNA*, the sequence-structure alignment program will be given. Following is formally describing the modified alignment algorithm, then the algorithm for generating isosteric motifs, the string searching algorithms implemented, and the modifications to *LocARNA* to include the matching of motifs. The results of the different parts of work are presented, and then follows are a conclusion and suggested future work.

# Chapter 2

# Background

## 2.1 Bioinformatics, DNA and RNA

In the last few decades, great advances in the field of molecular biology were made. In order to analyze the huge amount of new data, like the DNA sequences that bypass millions and billions of characters, the aid of the computer was needed. Bioinformatics, also known as computational biology, is concerned with the development of efficient algorithms, statistical analysis and mathematical modeling to store and analyze biological data.

Among the important problems of bioinformatics is the analysis of nucleic acids: DNA and RNA. Both DNA and RNA are polymers, which are composed of nucleotides. A nucleotide is a molecule consisting of a base, a ribose sugar (in DNA, deoxyribose), and a phosphate molecule. In DNA, we have four bases: adenine(C), cytosine(C), guanine(G) and thymine(T). In RNA, thymine is replaced by uracil(U). The adenine and uracil (and thymine) have 2 hydrogen bond sites whereas cytosine and guanine have 3 hydrogen bond sites.

Ribonucleic acid (RNA), which will be the main focus of the section, is believed to have existed before DNA, where it played both information storage and enzymatic role. Its extra hydroxyl group (OH) at the 2' position, gives it the ability to form more hydrogen bonds than DNA. In contrast to DNA, RNA is single stranded. It can perform Watson–Crick hydrogen bond pairs (A-U, C-G) and some weaker pairs, forming hairpin loops and more complex structures [2]. Unlike DNA that serves only for information storage, RNA can perform specific functions according to its complex 3D structure. Among the kinds of RNAs are messenger RNA (mRNA), transfer RNA (tRNA) and ribosomal RNA (rRNA). The structure of RNAs, similar to the one of proteins, is described on different levels. A brief explanation of these levels is presented here and more detailed explanation of some points is given later. Figure 2.1 illustrates the different levels of RNA structure.

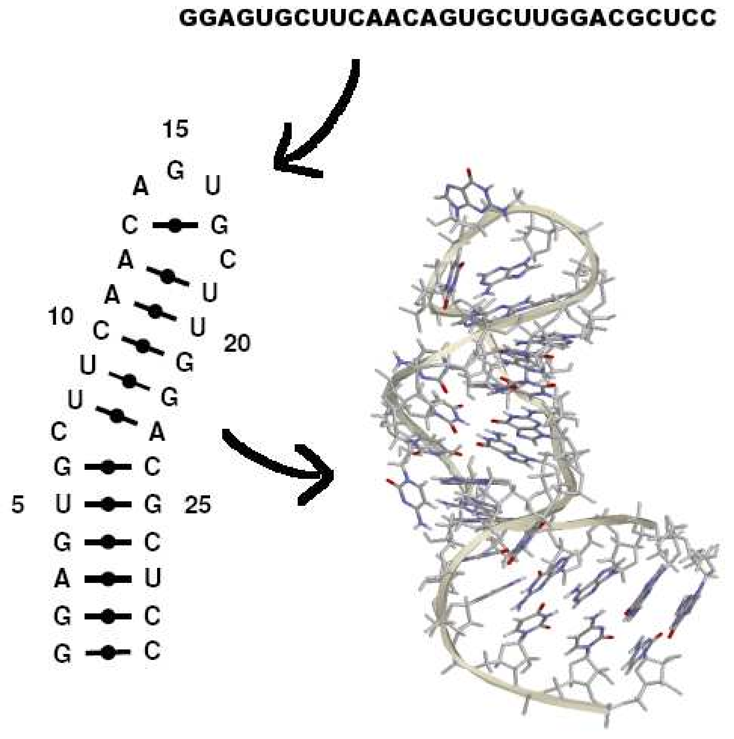


Figure 2.1: Structures of an RNA molecule. Top of the picture is an RNA molecule represented in its primary structure, left is the secondary structure, and right the tertiary structure. Taken from [3]

- **primary structure:** It describes the RNA on the sequence level, as an ordered list of nucleotides (bases) over the alphabet {A, C, G, U}. The sequence of bases is attached to a sugar–phosphate backbone. The primary representation does not describe the structure of RNA or its form in 3D.

- **secondary structure:** It describes the RNA on the structural level. RNA is single-stranded in its normal state, but due to hydrogen bonds, it folds into a functional shape by forming intermolecular base pairs among some of its bases. The set of these base pairs is known as the secondary structure of the RNA. The usual way of interaction between RNA bases is via Watson–Crick base pairing between the bases A with U and C with G. There are other non-Watson–Crick base pairs which form complex structures. These types of interaction will be explained later in details. An example of a Waston–Crick base pair between 'A' and 'U' bases is given in Figure 2.2. The molecular distance of the two bases in the bond is the distance between the two C1' atoms (represented by $\sim$ in the figure). This distance is known as the C1'–C1' distance. The secondary structure of RNA can be also predicted using the sequence. This problem is referred to as the RNA folding problem.

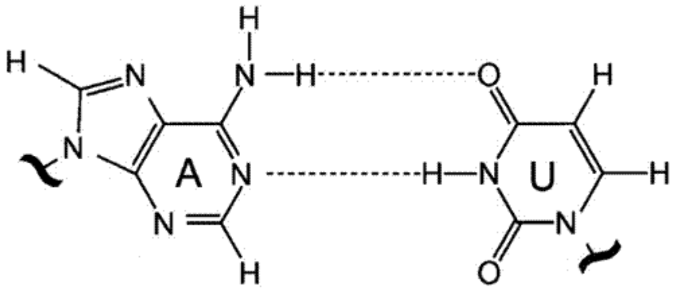


Figure 2.2: Waton–Crick base pair between 'A' and 'U' bases with C1'–C1' distance = 10.3 Å. Taken from [11], Figure 1

- **tertiary structure:** Under appropriate conditions, structured RNA molecules undergo a transition to a 3D fold in which the paired and unpaired are precisely organized in space. The tertiary structure is the level of organization relevant for biological function of structured RNA molecules. The secondary and tertiary structures are more reserved than primary structure during evolution. There are methods that determine the tertiary structure of RNA. One method relies on X-ray crystallography of single crystals of purified RNA molecules. Another one is nuclear magnetic resonance(NMR). [13]. However, the detection of the 3D structure of RNAs is a long, hard and expensive task. Only a limited number of RNAs have a known tertiary structure.

- **quaternary structure :** It is the arrangement of multiple folded RNAs in a multi-subunit complex.

## 2.2 RNA alignment

An alignment of DNAs, RNAs or proteins is a comparison of their sequences to detect and evaluate regions of similarity that may be a consequence of functional or evolutionary relationships. There are several kinds of alignments and different algorithms to solve each kind. In the following subsection, a description of the different kinds of alignments is presented.

### 2.2.1 Sequence alignment

In a sequence alignment, only the primary structure of the RNAs is taken into consideration. The alignment is done on the sequence level, ignoring the structure of the RNA. To align 2 RNA sequences, gaps are inserted such that the resulting strings have the same length and the score of the alignment is maximized according to a scoring function. An RNA sequence $S$ is a word over the alphabet $\Sigma = \{\text{A, C, G, U}\}$, $S[i]$ denotes the $i$th symbol of $S$. A scoring function $\sigma$ over the alphabet $\Sigma$ is a function $\sigma(x, y) \rightarrow \mathbb{R}$. An example of a scoring function is:

$$\sigma(x, y) = \begin{cases} 2 & \text{if } x = y \\ -1 & \text{otherwise} \end{cases}$$

The dynamic programming algorithm proposed by Needleman and Wunsch is the most famous algorithm to solve the sequence alignment problem. Let $A$ and $B$ be 2 RNA sequences with $|A| = n$ and $|B| = m$, we define a matrix $D_{n+1,m+1}$, and fill it with the following formulas: $D_{0,0} = 0$, $D_{0,j} = \sum_{k=1}^{j} \sigma(-, B_k)$, $D_{i,0} = \sum_{k=1}^{i} \sigma(A_k, -)$ and

$$\forall i, j > 0 : D_{i,j} = max \begin{cases} D_{i,j-1} + \sigma(-, B_j), \\ D_{i-1,j} + \sigma(A_i, -), \\ D_{i-1,j-1} + \sigma(A_i, B_j) \quad \text{if } A_i = B_j \end{cases}$$

After filling the matrix, a trace back method is used to determine the 2 strings $A'$ and $B'$ after alignment. An example of aligning 2 strings $A$=AGAUCGU and $B$=AGCAUG results in the alignment shown in Table 2.1, where the – symbol indicates a gap inserted in the sequence.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $A\prime$ | = | A | G | – | A | U | C | G | U |
| $B\prime$ | = | A | G | C | A | U | – | G | – |

Table 2.1: Example of sequence alignment

### 2.2.2 Sequence-structure alignment

In addition to sequence information, structure plays an important part in assessing the similarity of RNAs. The input of a sequence structure alignment is not only the sequence information, but also a set of arcs. An arc $a$ is a pair $(i, j) \in \mathbb{N} \times \mathbb{N}$, such that $i < j$. $i$ and $j$ are called the ends of $a$. A structure $P$ is a set of arcs, such that no end of an arc appears more than once in $P$. There are 2 kinds of structures, crossing and non crossing. We call two arcs $(i_1, i'_1)$, $(i_2, i'_2)$ crossing if and only if $i_1 < i_2 < i'_1 < i'_2 \vee i_2 < i_x 1 < i'_2 < i'_1$. A structure containing at least one pair of crossing arcs is called crossing, otherwise it is non-crossing. Finding the optimal alignment for 2 sequence-structures where both structures are crossing is an NP complete problem. If the structures are non-crossing, a dynamic programming solution exists for the problem.

If the 3D structure of an RNA is known, aligning it with another 3D structure is possible, but the problem is almost equal hard to the one of aligning crossing vs. crossing structures, which is NP complete [1]. An example of sequence structure alignment is given in Table 2.2, where every opening bracket ‘(’ and its corresponding closing bracket ‘)’ indicate a base pair.

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ( | ( | ( | ( | ( | ( | ( | . | . | . | ) | ) | . | ) | ) | ) | ) | ) |
| G | A | U | A | G | A | G | U | A | A | C | U | G | C | U | G | U | C |
| G | U | G | A | G | U | U | A | A | U | A | G | - | C | U | C | A | U |
| ( | ( | ( | ( | ( | ( | ( | . | . | . | ) | ) | - | ) | ) | ) | ) | ) |

Table 2.2: Example of sequence-structure alignment

### 2.2.3 Local vs. global alignment

Sometimes, only parts of the RNA sequences are conserved throughout evolution. Global alignments will give highly dissimilar sequences, whereas parts of these sequences are highly identical. Local alignment is introduced for this reason, where it tries to find the maximum score for aligning subsequences of RNA sequences. Common sequence structure features in two or several RNA molecules are often only spatially local, where possibly large parts of the molecule are dissimilar. Hence, local alignment in the sequence structure level is also important. For RNA, several types of locality are possible. If we define the local alignment as the best alignment of subsequences, we ignore completely the RNA structure. Hence, we require that the subsequences represent complete substructures (arc complete). That means that for an arc $(i, j)$, either $i$ and $j$ both take part in the local alignment or are both excluded. On the sequence and structure level, efficient algorithms solve the problem with the same complexity as the global alignment[1, 10]. An example of sequence structure alignment is given in table 2.3, where the $\sim$ symbol indicate that the corresponding character in the other sequence is excluded from the alignment.

```
.(((((((...)).))))))......∼∼∼∼∼
UGAUAGAGUAACUGCUGUCUUUAAA∼∼∼∼∼
∼GUGAGUUAAUAG–CUCAU∼∼∼∼∼∼AAUUG
∼(((((((...))–)))))∼∼∼∼∼∼.....
```

Table 2.3: Example of local sequence-structure alignment

### 2.2.4 Multiple alignment

Multiple alignment is an alignment for three or more DNA, RNA or protein sequences. In general, the sequences in theinput set are assumed to have an evolutionary relationship and are descendant from a common ancestor. Multiple alignment too can be done on the sequence level or sequence-structure level.

The complexity of the exact algorithm used to solve multiple alignment is exponential to the number of input sequences. Heuristics based on pairwise sequence-structure comparisons are used (e.g. progressive alignment)[9].

## 2.3 Motifs

Highly repeated RNA 3D motifs were observed in crystallized RNA molecules. These motifs are defined as a list of non-Watson–Crick base pairs. Isosteric motifs are ones that are different on the sequence level, but have the same 3D structure. The isosteric motifs, represented on the sequence level, can be detected in an alignment. In this section, the different types of non-Watson–Crick base pairs are presented, followed by a definition of RNA motifs and isostericity matrices.

### 2.3.1 Non-Watson–Crick base pairs

The most common RNA alignment programs present today are on the sequence–structure level. As discussed earlier, the prediction of the RNA tertiary structure is NP complete, and only a limited number of RNAs have a known tertiary structure. However, the function of the RNA highly depends on its 3D structure. The fundamental way of interaction between the bases of an RNA molecule is to form a Watson–Crick base pair. The rapid progress of crystallographic studies of RNAs revealed other modes of interaction, which are referred to as non-Watson–Crick base pairs. They were classified in twelve basic geometric families, based on the interacting edges of the bases and glycosidic bond orientation. The interacting edge of a base is either the Watson–Crick edge, the

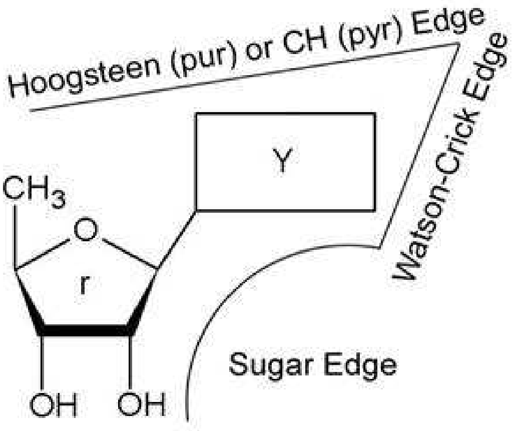
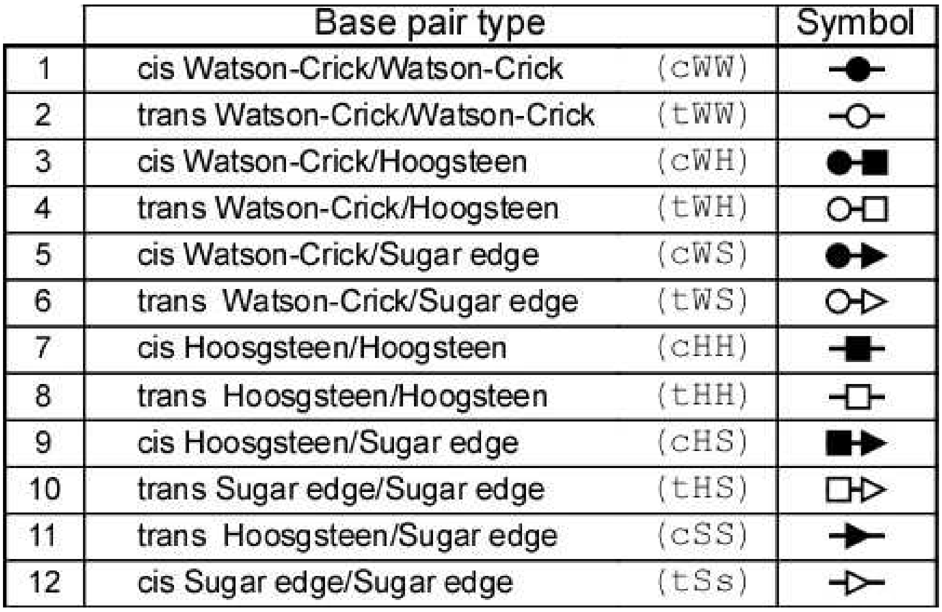


Figure 2.3: Edges of an RNA molecule. Taken from [12], Figure 1.

Hoogsteen edge or the sugar edge, shown in Figure2.3. Bases can interact in either of two orientations with respect to the glycosic bonds, cis or trans, relative to the hydrogen bonds.

The nomenclature described in [4] will be followed throughout the document. In this nomenclature, the Watson–Crick edge is described by a circle, the Hoogsteen edge by a square and the sugar edge by a triangle. Solid symbols indicate cis base pairs and open symbols trans base pairs. The symbols used in the nomenclature are given in Figure 2.4.



| | Base pair type | | Symbol |
|---|---|---|---|
| 1 | cis Watson-Crick/Watson-Crick | (cWW) | -●- |
| 2 | trans Watson-Crick/Watson-Crick | (tWW) | -○- |
| 3 | cis Watson-Crick/Hoogsteen | (cWH) | ●-■ |
| 4 | trans Watson-Crick/Hoogsteen | (tWH) | ○-□ |
| 5 | cis Watson-Crick/Sugar edge | (cWS) | ●-▶ |
| 6 | trans Watson-Crick/Sugar edge | (tWS) | ○-▷ |
| 7 | cis Hoosgsteen/Hoogsteen | (cHH) | -■- |
| 8 | trans Hoosgsteen/Hoogsteen | (tHH) | -□- |
| 9 | cis Hoosgsteen/Sugar edge | (cHS) | ■-▶ |
| 10 | trans Sugar edge/Sugar edge | (tHS) | □-▷ |
| 11 | trans Hoosgsteen/Sugar edge | (cSS) | -▶- |
| 12 | cis Sugar edge/Sugar edge | (tSs) | -▷- |

Figure 2.4: The nomenclature followed in the document proposed by [4].

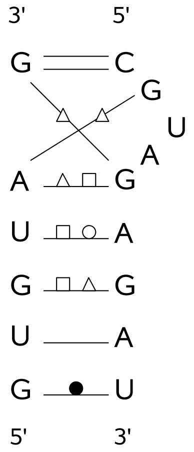


Figure 2.5: A motif with the interactive base pairs shown

### 2.3.2 RNA structural motifs

The non-Watson–Crick base pairs combine to form more complex local RNA motifs, which are defined operationally as "ordered arrays of non Watson–Crick base pairs forming distinctive folding"[6]. Due to the strong and complicated bonds found in the structural motifs, they are highly conserved throughout evolution. These motifs were noticed with the observation of the crystallized RNAs within the same families. It was found that these motifs were highly repeated in the RNA molecules.

The sequence signature of a motif is defined as a set of nucleotides that fold to form the same 3D motif. Given the sequence signature of the motifs, one could easily detect them in a sequence alignment software. The detection of the structural motifs on the sequence level can give a very strong guidance for the alignments. An example of a motif is given in Figure 2.5, with the different interactions between the bases shown.

### 2.3.3 Isostericity matrices

The sequence signature of motifs is not always fully conserved. As observed in the crystallized RNA molecules, some bases change in the motifs. However, the 3D structure of the motif always remains the same.

Base pairs that can be substituted in a motif without changing the molecular distance between the bases are called isosteric. To identify isosteric base pairs, we begin with the base pairs that belong to the same geometric family because they share the same relative orientations of the glycosidic bonds of their interacting bases and so meet the first criterion of isostericity. However, due to the size difference between the types of bases, not all the base pairs in the same geometric family have the same C1'–C1' distances separating the interacting bases. The C1'–C1' is therefore is the second criterion for isostericity. Thus, base pairs that are in the same geometric family and have roughly equal C1'–C1' distances and are hydrogen bonded between equivalent atomic positions form together an isosteric group of base pairs. The base pairs are presented in 4 × 4 isostericity matrices, one for each geometric family. Isostericity matrices therefore indicate which base pairs in each geometric family can substitute for each other without changing the 3D structure of the motif to which they belong. The isostericity matrix for the cis Watson–Crick - Hoogsteen base pairs is given in table 2.4

| cis | A | C | G | U |
|---|---|---|---|---|
| A | $I_4$ |  | $I_4$ |  |
| C | $I_2$ | $I_1$ | $I_2$ |  |
| G |  |  | $I_5$ | $I_4$ |
| U | $I_1$ |  | $I_3$ | $I_2$ |

Table 2.4: Isostericity matrix of the cis Watson–Crick - Hoogsteen base pairs, where base pairs having the same index are isosteric

# Chapter 3

# Local sequence-structure aligment tool: *LocARNA*

*LocARNA*[14](**Loc**al **A**lignment of **RNA**), is a tool for local alignment of RNAs. It is a Sankoff-style algorithm, but further introduces local alignment. It is implemented in C++.

The Sankoff algorithm[8] provides a general solution to the problem of simultaneously computing an alignment and the common secondary structure of the two aligned sequences. The algorithm requires $O(n^6)$ time complexity and $O(n^4)$ space complexity, where $n$ is the length of the RNA sequences to be aligned.

*LocARNA* assumes that a structure model for the 2 input sequences is already known and given in the form of weights for the individual base pairs. The structural information is computed using McCaskill's algorithm, implemented in *RNAfold*. This algorithm computes a matrix of pair probabilities based on a complete energy model of RNAs.

Consider two sequences $A$ and $B$ with associated base pair probabilities matrices $P^A$ and $P^B$, respectively. The goal is to compute a sequence alignment $\mathbf{A}$ of $A$ and $B$ together with the secondary structure $\mathbf{S}$ on $\mathbf{A}$. $\mathbf{A}$ consists of a set of (mis)matches written as pairs $(i, k)$, where $i$ is the position in $A$, and $k$ is the position in $B$. The consensus secondary structure $\mathbf{S}$ for an alignment $\mathbf{A}$ consists of a set of quadruples $(ij; kl)$, where $(i, k) \in \mathbf{A}$

and $(j, l) \in \mathbf{A}$ are two matches in $\mathbf{A}$, $(i, j)$ is a base pair on sequence $A$ and $(k, l)$ is a base pair on sequence $B$. Furthermore, $\mathbf{A}_s$ denotes the pairs of the alignment that do not contribute in any base pairs of the alignment, i.e., if $(i, k) \in \mathbf{A}_s$, then there exists no pair $(j, l)$ such that $(ij, kl) \in \mathbf{S}$. The goal is to determine the pair $(\mathbf{A}, \mathbf{S})$ that maximizes the score function:

$$\sum_{(ij,kl)\in\mathbf{S}} (\psi_{ij}^A + \psi_{kl}^B) + \sum_{(i,k)\in\mathbf{A}_s} \sigma(A_i, B_k) - N_{gap}\nu,$$

where $\psi_{ij}^A$ and $\psi_{kl}^B$ are base pair scores (explained below), $\sigma : \{A, C, G, U\}^2 \to \mathbb{R}$ is the similarity score for (mis)matches, $\nu$ is the gap score parameter and $N_{gap}$ is the number of insertions and deletions in alignment $\mathbf{A}$.

The base pair scores are derived from the base pair probability matrix of the two individual sequences.

$$\psi_{ij} = \begin{cases} cP_{ij} & \text{if } P_{ij} \geq p^* \\ -\infty & \text{otherwise} \end{cases}$$

where $P_{ij}$ is the pairing probability, $c$ is a normalizing factor to make it easier to balance the sequence score against the structure score, and $p^*$ is the cut-off probability below which the arcs are ignored. Formally, it is expressed by giving a score of $-\infty$ as score.

The implemented algorithm has the usual Sankoff-Style form. $D_{ij;kl}$ is defined as the maximal similarity score of an alignment for the subsequences $A[i..j]$ and $B[k..l]$ with the additional condition that $(ij; kl)$ is part of the consensus secondary structure. To profit from the fact that a base pair with probability below $p^*$ is ignored, only $D_{ij;kl}$ are calculated and stored for significant base pairs. The calculation of $D_{ij;kl}$ is computed by fixing $i$ and $k$ and varying $j$ and $l$. The calculation of a $D$ entry for $i$ and $k$ is done through an auxiliary matrix $M$, where the entries $M_{ij;kl}$ are the optimal similarity scores of subsequences $A[i+1..j]$ and $B[k+1..l]$, leading to a computational time of $O(n^2)$. The dynamic programming algorithm is:

$$M_{ij;kl} = \max \begin{cases} M_{ij-1;kl-1} + \sigma(A_j, B_l) \\ M_{ij-1;kl} + \nu \\ M_{ij;kl-1} + \nu \\ max_{j'l'}(M_{ij'-1;kl'-1} + D_{j'j;l'l}) \end{cases}$$

$$D_{ij;kl} = M_{ij-1;kl-1} + \psi^A_{ij} + \psi^B_{kl}$$

The computation of the $D$ for $i$ and $k$ depends only on the matrix $M$. We only need to store the entries of $M$ for the current $i$ and $k$ values, i.e. $O(n^2)$ entries. With a constant $p^*$, every base in $A$ or $B$ can take only part in at most $1/p^*$ many base pairs, thus $O(1)$ base pairs for every base. Hence, there is a total of $m = O(n)$ base pairs for every sequence. This makes a total of $O(m^2)$ in $D$. The total space complexity is $O(m^2 + n^2)$, and time complexity is $O(n^2(n^2 + m^2))$.

The score of the global alignment is $M_{0|A|;0|B|}$. Modifications to the algorithm are made to support calculating the best local alignment [14], however these lengthy modifications are not described here.

# Chapter 4

# Extension of *LocARNA* by the detection of 3D structural motifs

The main task of this project is to extend *LocARNA* with the detection of 3D structural motifs. The idea was to implement a motif detector that is independent from the *LocARNA* engine, and run it as a preprocessing step before running the engine. The output of the motif detector is a list containing all the motifs found in the 2 RNA sequences that are to be compared. All this is done as a preprocessing step, in order not to increase the complexity of the main engine of *LocARNA*. All of the preprocessing steps are of negligible complexity compared to the complexity of the main engine.

The list of motifs is generated using the search algorithm described in Subsections 4.2.1 and 4.2.2. A motif is represented as two ranges(substrings) of an RNA sequence, $([i..j],[k..l])$, and is assumed to be surrounded by base pairs (see Figure 4.1). Let $M^A$ and $M^B$ be the two lists of motifs found in sequences $A$ and $B$, respectively. Each element in the list will hold information about each motif such as the position of its 2 ranges in the RNA sequence and an identifier to it.

The definition of an optimal alignment is modified to include a score for aligning two motifs. The motifs alignment **M** for an alignment **A** is a set of four ranges $([r_A^1..r_A^{1'}],[r_A^2..r_A^{2'}];[r_B^1..r_B^{1'}],[r_B^2..r_B^{2'}])$, where $([r_A^1..r_A^{1'}],[r_A^2..r_A^{2'}]) \in M^A$, $([r_B^1..r_B^{1'}],[r_B^2..r_B^{2'}]) \in$

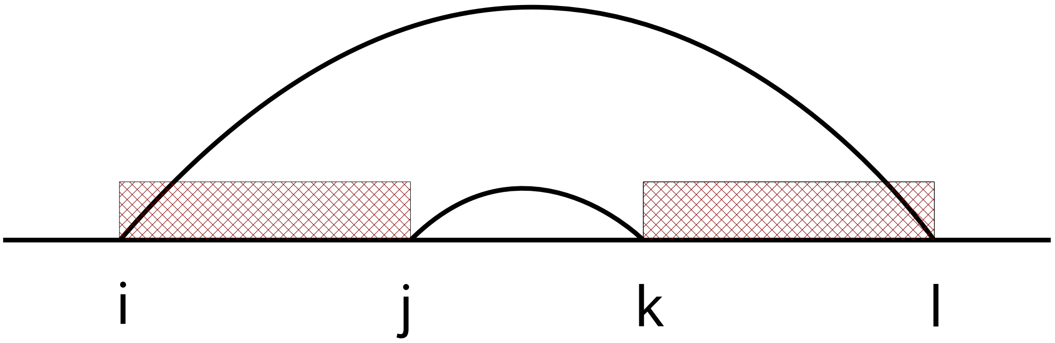


Figure 4.1: Motif surrounded by base pairs

$M^B$, $P^A_{r^1_A r^{2'}_A} \geq p^*$, $P^A_{r^{1'}_A r^2_A} \geq p^*$, $P^B_{r^1_B r^{2'}_B]} \geq p^*$ and $P^B_{r^{1'}_B r^2_B} \geq p^*$. The single-stranded part of the alignment $\mathbf{A}_s$ is redefined such that if $(i,k) \in \mathbf{A}_s$ then there exists no pair $(j,l)$ such that $(ij,kl) \in \mathbf{S}$ and there exists no motif $([r^1_A..r^{1'}_A],[r^2_A..r^{2'}_A];[r^1_B..r^{1'}_B],[r^2_B..r^{2'}_B]) \in \mathbf{M}$ such that $r^1_A \leq i \leq r^{1'}_A \vee r^2_A \leq i \leq r^{2'}_A \vee r^1_B \leq k \leq r^{1'}_B \vee r^2_B \leq k \leq r^{2'}_B$. An extra condition is given for the set of consensus secondary structure $\mathbf{S}$, that for every base pair $(ij;kl) \in \mathbf{S}$, the last condition for a pair in $\mathbf{A}_s$ apply for the pairs $(i,j)$ and $(k,l)$.

The goal is to determine the triplet $(\mathbf{A},\mathbf{S},\mathbf{M})$ that maximizes the score function:

$$\sum_{(R_A R_B)\in\mathbf{M}} \eta(R_A,R_B) + \sum_{(ij,kl)\in\mathbf{S}} (\psi^A_{ij} + \psi^B_{kl}) + \sum_{(i,k)\in\mathbf{A}_s} \sigma(A_i,B_k) - N_{gap}\nu,$$

where $R_A$ and $R_B$ represent two motifs found in $A$ and $B$, respectively. $\eta(R_A,R_B)$ is the motifs score,

$$\eta(R_A,R_B) = cd^{e_1+e_2}$$

where $c$ is the scoring constant, $d < 1$ is a constant for decreasing the score in case of errors in the motifs, $e_1$ and $e_2$ are the number of errors found in motifs $R_A$ and $R_B$ respectively. Errors can be found in motifs in case of the usage of an approximate search algorithm (see Subsection 4.2.2). If an exact algorithm is used (see Subection 4.2.1), $e_1$ and $e_2$ will be equal to 0, and the score will be equal to $c$.

The dynamic programming algorithm will be modified to include motifs:

$$M_{ij;kl} = \max \begin{cases} M_{ij-1;kl-1} + \sigma(A_j, B_l) \\ M_{ij-1;kl} + \nu \\ M_{ij;kl-1} + \nu \\ max_{j'l'}(M_{ij'-1;kl'-1} + D_{j'j;l'l}) \end{cases}$$

$$D_{ij;kl} = \max \begin{cases} M_{ij-1;kl-1} + \psi^A_{ij} + \psi^B_{kl} \\ max_{\forall i'j'k'l'}(\eta([i..i'],[j'..j];[k..k'],[l'..l]) + D(i'j';k'l')) \end{cases}$$

Before starting the dynamic programming algorithm, a refinement is done to the two lists of motifs $M^A$ and $M^B$. For a motif $([i..j][k..l])$, if $P_{il} < p^*$ or $P_{jk} < p^*$, then the motif will never be aligned, and thus removed from the list. Given the fact that all the motifs are considered isosteric, then having several motifs with the same surrounding arcs is useless and only one motif is kept in the list. If the motifs can contain errors, then the one with the least number of errors is kept.

The modification of the dynamic programming algorithm is done in the calculation of an entry in the $D$ matrix. An entry $D_{ij;kl}$ gets the maximum score of aligning the arcs $(i, j)$ with $(k, l)$, and aligning two motifs surrounded by arcs $(i, j)$ and $(k, l)$, respectively. Assuming that all the motifs are of equal length, which is the case for isosteric motifs, only one motif can be surrounded by a specific arc. The length of a motif can change in case of a constant number of allowed errors (insertions or deletions), but the number of motifs surrounded by a specific arc remains constant. The maximum size of the list motifs is $O(m^2)$, since the number of motifs surrounded by an arc is constant. The time and complexity of the modified algorithm remains the same as the original one.

## 4.1 Search for new 3D structure motifs

This section describes the algorithm used to combinatorially generate all the isosteric motifs to one original motif. The input is the isostericity matrices given in [5] and one motif where the types of bonds between the bases are given. As explained earlier, isosteric bonds are the ones that can be substituted by one another without making a difference in the 3D structure of the RNA. In the observed RNA motifs, the type of the bonds is known, so there is no problem in detecting all the isosteric bonds to the one observed using the isostericity matrices.

A simple recursive algorithm can be used to generate all the possible motifs where each bond in the new motifs is isosteric to the corresponding one in the original motif. A pseudo–code of the algorithm is:

```
procedure GENERATE_ALL(sequence)
    if all bonds are marked as visited then
        print(sequence)
        return
    end if
    $b \leftarrow$ not visited bond
    mark $b$ as visited
    for all isosteric bond to $b$ do                              ▷ Including $b$
        $temp \leftarrow sequence$
        $temp \leftarrow temp$ with characters of $b$ replaced by characters of the new bond
        generate_all($temp$)
    end for
end procedure
```

The first call to the procedure is *generate_all*(original RNA sequence), with all the bonds marked as not visited. In an RNA motif, some characters are present in more than one base pair. This case is not handled in the algorithm. A slight modification is added to the recursion: before replacing each character of the bond, check if it has already been visited. If it is, then compare it with the character of the current bond being checked. If they are the same, continue with the normal recursion. If not, then discard this sequence. If the character has not been visited, do the normal procedure. All the characters are initialized as non-visited.

Some of the characters of the motif are not interacting in any of the bonds. These characters are considered as wildcards, which means that substituting them with any character will still result in the same motif. The wildcards are represented by the character '?' in the list of motifs.

## 4.2 String searching algorithms

The search for a pattern in a text, or simply the string searching problem is a classical problem of computer science. Formally, the string searching algorithm is the search for a string $P = p_0p_1...p_m$ inside a large string $T = t_0t_1...t_n$, both sequences are characters from an alphabet $\Sigma$. The alphabet in our case is the RNA characters, $\Sigma = \{$A, C, G, U$\}$. We want to find all occurrences of P in T; namely, we are searching for the set of starting positions $F = \{i|0 \leq i \leq n - m$ such that $t_it_{i+1}...t_{i+m-1} = P\}$
There are a lot of famous algorithms for string searching, like the Boyer-Moore algorithm, the Knuth Morris Pratt algorithm, the deterministic finite automata and the Bitap algorithm. To analyze the complexity of a string searching algorithm, 2 factors have to be taken in consideration: the preprocessing time of the algorithm and the matching time. As both phases are considered as a preprocessing step for *LocARNA*, the overall complexity of the searching algorithm becomes $O$(preprocessing time + matching time). Two string searching algorithms were implemented to search in the RNA sequence for motifs, the first one constructs a tree of patterns and searches for all the patterns at once, and the other is the Bitap algorithm which searches for the patterns one by one but allows searching for a pattern with errors.

### 4.2.1 Fast motif search algorithm

The fast motif search is called so because it searches for all the patterns in one step. The structure implemented to do this search is a quadtree, which means that a node can have at most 4 children. I will call the tree in this context the patterns tree, because it holds all the patterns to be searched for. An edge in the patterns tree represents a transition by an RNA character from the parent to the child. For each node there are 4 possible children, one for each RNA character A, C, G, U. If a node $N$ with a depth $d$ in the tree is being visited, that means that the first $d$ characters of $T$ were matched with all the transition character from the root till $N$.

In the preprocessing step of this search algorithm, the patterns tree is constructed. A structure Node is used in the code to represent node in the tree. It has 2 attributes:

- a list of four Nodes (the children): *transition*. It represents the transition from this Node with respect to the four allowed characters: {A, C, G, U}. The 4 Nodes are initialized as null in the Node structure constructor.
- a list of numbers: *accepted*. It is initialized as an empty list. Every number $i$ of *accepted* means that motif with index $i$ ends with this node.

The construction of the patterns tree goes as follows:

```
procedure CONSTRUCT PATTERNS TREE(Pattern[] $p$)
    Node $root \leftarrow$ new Node
    for all Pattern $i : p$ do
        Node $current \leftarrow root$
        for all character $c : i$ do
            if $current$.transition[c] = null then
                $current$.transition[c] $\leftarrow$ new Node
            end if
            $current \leftarrow current$.transition[c]
            if $c$ is the last character of $i$ then
                $current$.accepted.add(index of pattern $i$)
            end if
        end for
    end for
end procedure
```

At the beginning of the algorithm, only the root is constructed. The addition of a pattern to the search tree is done as following: A node *current* is used to represent the current node being visited. For all the characters of the pattern, do: If the child of the node *current* which corresponds to the transition to the current character doesn't already exist, create it. Then let the node *current* point to it. If the current character of the pattern is the last one, add to the accepted list of this node (*current*) the index of the pattern being added. Figure 4.2 illustrates the case of adding a pattern GUGA to a tree already containing the pattern GUAG. The time complexity is: $O(nm)$, where $n$ is the number of patterns and $m$ is the length of a pattern.

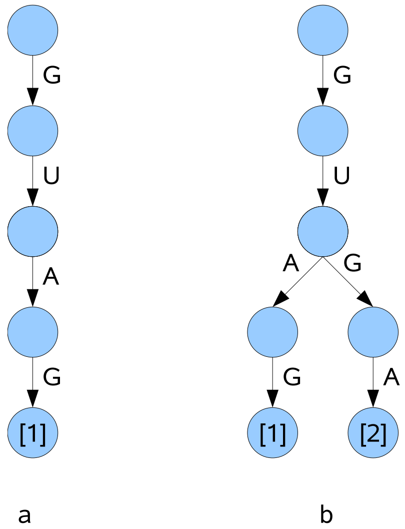


Figure 4.2: a) A tree with the pattern GUAG, b) Tree (a) with pattern GUGA added.

Since a motif is composed two ranges as explained earlier, the search for the two ranges must be independent. Each range of a motif is considered as an independent pattern to be searched for. The construction of the patterns tree can include both ranges of the motifs, or two pattern trees are constructed, one for the first range and the other of the second range of the motifs. Both choices are equivalent in complexity, the second choice is chosen for slight performance advantages.

After the preprocessing step comes the second part of the searching algorithm: the actual search in the text. Given a text $T = t_0 t_1 ... t_n$, start with the root node of the patterns tree. The result should be a list $res$, of size equal to the number of motifs. Each element of $res$ is also a list of integers, indicating the starting indices of this motif found in $T$. With every character $t_i$, make a transition of the current node to the child with corresponds to the transition $t_i$. If the node doesn't exist, stop the search. Else, append the elements of the accepted list of the current node to the list of found patterns. These steps will find all the patterns that begin with character $t_0$ of the text. To find the patterns beginning with any character of the text, the search operation described above should be done with all the suffixes of $T$.

```
procedure SEARCH TEXT(Node $root$, string $T$)
    list<list<integer>> $result$
```

```
3:     for all Suffix $i$ of text do
4:         Node $current \leftarrow root$
5:         for all Character $j$ of $i$ do
6:             $current \leftarrow current$.transition[$j$]
7:             if $current$ = null then
8:                 break
9:             end if
10:            for all motif $k \in current$.accepted do
11:                result[$k$].add($i$)
12:            end for
13:        end for
14:    end for
15:    return $result$
16: end procedure
```

,

The for loop in line 3 is executed for all the suffixes of $T$, which is equivalent to $|T|$ times. The loop in line 5 is executed for the maximum of the length of the suffix of $T$, and the depth of the patterns tree which is equal to the length of the patterns. In the normal case, the length of the patterns is much less than the length of the suffix of $T$, so we will only consider this case. The time complexity of the algorithm becomes $O(nm+l)$, where $n$ is the length of $T$, $m$ is the length of one pattern, and $l$ is the number of the overall motifs found.

Now comes the last part of the searching algorithm, the construction of the motif list that will be used in *LocARNA*. The construction of the patterns tree and the search for patterns in the text is done independently for first range of all motifs, then for the second range. For every motif, there exist two lists, the occurrences of its two ranges in $T$. To get a single list of allowed motifs, every element of the list of the first range is compared to all the elements of the second range. If the occurrence of the first range occurs in $T$ before the second range, then this is a valid motif. The pseudo code is:

```
for all motif $m$ do
    for all range $i$: found first range of $m$ do
        for all range $j$: found second range of $m$ do
            if $i$ occurs before $j$ in $text$ then
                add new motif($i, j$) to the list of found motifs
            end if
        end for
    end for
end for
return list of found motifs
```

After analyzing the complexity of every piece of code, the complexity of the whole algorithm can be computed. The complexity of the preprocessing step is $O(nm)$, where $n$ is the number of patterns and $m$ is the length of a pattern. The complexity of the actual search is $O(km+l)$, where $k$ is the length of $T$, and $l$ is the total number of motifs found. The total time complexity is $O(nm + km + l)$. The actual length of one pattern is between 6 and 10 characters, which can be considered as a constant. This complexity of the search algorithm is considered negligable when comparing it to the complexity of *LocARNA*.

### 4.2.2 Bitap's algorithm

Bitap's algorithm, also known as shift-or, shift-and or Baeza-Yates–Gonnet algorithm, is an algorithms that supports searching for approximate patterns. It was developed by Sun Wu and Udi Manber in 1992, based on an exact string matching scheme developed by Baeza-Yates and Gonnet[15].

Approximate or fuzzy searching is the technique of finding strings(patterns) that are close to a substring of a string(text). The closeness is measured by the Levenshtein distance. The need of approximate searching is strong in the case of motif search because the motifs are not always fully conserved throughout evolution. The changes that can occur in a motif are insertion, deletion or substitution of a base. In addition, the bitap algorithm allows wildcards, which are character that may be substituted by any other character without adding a cost. The algorithm is based on bit-wise operations, which are very fast.

The simple case of exact matching is described here. Let $R[m{+}1][n{+}1]$ be a 2-dimensional bit array, where $m$ is the length of the pattern and $n$ the length of the text. Initially, $R[i][0] = 1$ for $0 \leq i \leq m$ and $R[i][j] = 0$ for $0 \leq i \leq m \wedge 0 < j \leq n$

The table is filled by the following equation for $1 \leq i \leq m$ and $1 \leq j \leq n$

$$R[i][j] = \begin{cases} 1 & \text{if } R[i-1][j-1] = 1 \text{ and } p_{i-1} = t_{j-1} \\ 0 & \text{otherwise} \end{cases}$$

If $R[m][i]$ is equal to 1, we output a match beginning at index $j - m + 1$. This transition, which we have to compute once for every character, is computationally expensive. However, a trick improves the algorithm: For every character $c$ of the alphabet $\Sigma$, we construct a bit array $S_c$ such that $S_c[i] = 1$ iff $p_i = c$. To include the search for wildcards, we make a simple modification: $S_c[i] = 1$ iff $p_i = c \vee p_i =?$. The bit array $S$ can be also be treated as an integer, and setting the $i$th character to 1 is equivalent to $S_c = S_c|(1 << i)$, where the — symbol is the bitwise OR operations, and $<<$ is the left shift operation. The construction of the bit arrays, which will be called as *pattern_masks* is:

```
int $pattern\_masks[|\Sigma|]$
for all $i = 0..$ length of pattern do
    $c \leftarrow pattern[i]$
    if $c =?$ then
        for all $j \in \Sigma$ do
            $pattern\_masks[j] = pattern\_masks[j]|(1 << i)$
        end for
    else
        $pattern\_masks[c] = pattern\_masks[c]|(1 << i)$
    end if
end for
return $pattern\_masks$
```

An example of an exact match with the associated pattern masks is given in table 4.1. Table 4.2 shows an example of exact match and match with at most one insertion.

| | | A | G | U | C | C | A | C | G | U | U | A | C | A | A | C | G | U |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | |
| G | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| U | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| ? | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| | | | | | | | a | | | | | | | | | b | | |

Table 4.1: Example of exact matching and the corresponding pattern masks

|  |  | A | G | U | C | C | A | C | G | U | U | A | C | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| G | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| U | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| ? | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

a

|  | A | G | U | C | C | A | C | G | U | U | A | C | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |

b

Table 4.2: a)Example of exact matching and b) matching with at most one insertion(b)

The worst-case time complexity of the above algorithm is $O(m|\Sigma|)$, where $m$ is the length of the pattern. After the construction of the *pattern_masks* array, if we consider the 2 dimensional bit array $R$ as a single integer array, the transition from $R[i]$ to $R[i+1]$ becomes: $R[i+1] = (R[i]\&pattern\_masks[p_i]) << 1$, $R[i+1] = R[i+1]|1$. The complexity of the algorithm is reduced to $O(n)$, where $n$ is the length of the text.
The advantage of the Bitap algorithm is that it can be easily modified to include the search for approximate patterns. An error in the text is an insertion, deletion or substitution. Each of the cases can be handled easily with the algorithm. We will begin by investigating the case of handling $k$ insertion: We define the $R$ and *pattern_masks* arrays as before, but $k$ new arrays are needed for detecting a match with 1 to $k$ insertions. We denote these new arrays $R_d$, where $0 \leq d \leq k$ is the number of errors, $R_0 = R$ is the exact matches array. The transition for the $R_0$ array is the same as before. Only the transitions for $R_d$ arrays are needed. The transition to $R_d[i+1]$ indicated a match in position $j+1$(suppose we are working with the old bit arrays) in 2 cases:

1. There is a match with $d-1$ insertions in position $i+1$. In this case, inserting the character $text_i$ will create a match with at most $d$ insertions(Whether $t_i$ matches $p_j$ or not).

2. There is a match with $d$ insertions in position $i$, and $t_i = p_j$.

In pseudo code, this transition will look like:

$R_d[i+1] = R_{d-1}[i]$

$R_d[i+1] = R_d[i+1]|((R_d[i]\&pattern\_masks[t_i]) << 1)$

$R_d[i+1] = R_d[i+1]|1$

Now consider the case of deletions. Taking the same assumptions as before, the transition to $R_d[i+1]$ indicated a match in position $j+1$ in 2 cases:

1. There is a match with $d-1$ deletions in position $i+1$. In this case, deleting the character $t_i$ will create a match with at most $d$ deletions(Whether $t_i$ matches $p_j$ or not).

2. There is a match with $d$ deletions in position $i$, and $t_i = p_j$.

In pseudo code, this transition will look like:

```
$R_d[i+1] = (R_{d-1}[i+1] << 1)$
$R_d[i+1]| = R_d[i+1]|((R_d[i]\&pattern\_masks[t_i]) << 1)$
$R_d[i+1] = R_d[i+1]|1$
```

The final case is the one of the substitutions. Again, the transition to $R_d[i+1]$ indicated a match in position $j+1$ in 2 cases:

1. There is a match with $d-1$ substitutions in position $i$. In this case, substituting the character $t_i$ will create a match with at most $d$ substitutions(Whether $t_i$ matches $p_j$ or not).

2. There is a match with $d$ deletions in position $i$, and $t_i = p_j$.

In pseudo code, this transition will look like:

```
$R_d[i+1] = (R_{d-1}[i] << 1)$
$R_d[i+1]| = R_d[i+1]|((R_d[i]\&pattern\_masks[t_i]) << 1)$
$R_d[i+1] = R_d[i+1]|1$
```

The implemented Bitap algorithm can only work with 1 pattern. There are suggested modifications by [15] to make the algorithm detect multiple patterns, but they are hard to implement and will not greatly affect the complexity. The time complexity of the

Bitap algorithm with the detection up to $k$ errors is $O(nk)$ to fill the $n \times k$ array, each cell requires $O(1)$ time ($n$ is the length of the text). If we repeat the algorithm for all the patterns, the time complexity becomes $\Theta(nkt)$, where t is the number of patterns. Clearly, the complexity of the matching phase is much higher than the preprocessing phase, so the effective time complexity of the algorithm is $O(nkt)$.

When matching 2 motifs in *LocARNA*, previously there was a constant score giving to the match. With the errors introduced, matching motifs with errors must be given a less score than matching without errors or with fewer errors. If the constant score given to matching motifs is $c$, the score to matching 2 motifs, one with $e_1$ errors and the other with $e_2$ errors is: $score = cd^{e_1+e_2}$, $d < 1$.

# Chapter 5

# Results

The actual data that was available during the implementation of the algorithms presented in Chapter 4 are from the multiple alignment done by hand by Eric Westhof's team in Strasbourg. This alignment included 799 RNA sequences from the ribosomal 16S family. Every sequence had an average length of 1500 characters. In each sequence one motif was observed. After the analysis of the data, a total of 27 different motifs were discovered. A motif was considered different if it had at least 1 character of difference from the other motifs.

One motif of the 27 different motifs had occurred in 388 RNA sequences, and from it were generated all the other isosteric motifs. The list of all the interactions within the motif was also given. The isostericity matrices described by [5] were used together with the motif shown in Figure 2.5 to generate the motif signature. The algorithm used is the one described in Subsection 4.1. The algorithm was implemented using Java. Executing the program gave an output of 1536 isosteric motifs.

Comparing the list of isosteric motifs to the ones found in the global alignment, only 8 motifs were found out of the 27, the 8 motifs occurred in 663 out of the 799 sequences ($\approx$ 83%). Three motifs with a total of 92 occurrences ($\approx$ 11.5%) were missing because of a change in the characters not involved in any interactions. The remaining 16 motifs which occur $\approx$ 5.5%, were not found because of substitutions or insertions in the motifs.

The first searching algorithm 4.2.1, which is an exact searching algorithm, detected only the exact motifs stated above. It doesn't allow the use of wildcards or errors in the patterns(motifs). Testing the algorithm on 2 random sequences from the ribosomal 16S family gave a total of 1 motif in each sequence in a duration of 10 milliseconds with the refinement of the motifs. With the refinements of the motifs, the program detected 4 motifs in each sequence, which means that 1 motif is correct and the 3 others were wrong and were eliminated by the refinement of motifs.

The detection of the motifs was tested again with the bitap algorithm 4.2.2, which is an approximate algorithm. A total of 1 substitution, 1 insertion and 0 deletions were allowed in each range of the motifs, allowing a maximum of 2 errors within 1 motif. The wildcards don't count as an error. Searching for the 27 motifs found in the global alignment, the results were:

11 motifs of 27 occurring 754 times in 799 sequences ($\approx 94.4\%$) were exact matches.

6 motifs occurring 17 times ($\approx 2.1\%$) were found with 1 error.

9 motifs occurring 21 times ($\approx 2.6\%$) were found with 2 errors.

1 motif occurring once ($\approx 0.1\%$) was not found.

Table 5.1 shows all the motifs with the number of occurences in the multiple alignment and number of errors with respect to the closest motif in the isostericity matrix.

Running bitap's algorithm on 2 sequences from the multiple alignment gave 3 motifs in each sequence with the refinement of the motifs. Without refinement, an average of 16000 motifs was found in the 2 sequences!

Finally, a part of the output of a successful run of *LocARNA* with the exact searching algorithm and with refinement of the motifs is presented in Figure 5.2, where all the symbols are identical to the ones used in Table 2.2 (page 9), the square brackets [ ], and the $*$ symbol indicate the motifs.

| Motif | Occurences | number of errors |
|---|---|---|
| GUGUAG CGUAGAGAU | 388 | 0 |
| GUGUAG CGUAGAUAU | 251 | 0 |
| GUGUAG CGCAGAGAU | 49 | 0 |
| GUGUAG CGCAGAUAU | 41 | 0 |
| GUGUAG CUUAGAUAU | 16 | 0 |
| GUGGAG CGUAGAUAU | 10 | 1 |
| GUGUAA UGUAGAUAU | 6 | 2 |
| GUGUAG CAUAGAUAU | 5 | 0 |
| GAGUAG CGCAGAUAC | 4 | 2 |
| GUGUAG UGUAGAUAU | 4 | 2 |
| GUGUAG CGUAGAUAC | 3 | 1 |
| GUGUAG CGUAAAUAU | 3 | 0 |
| GUGUAG CGUARAUAU | 2 | 2 |
| GUGUAG CGUUGAUAU | 2 | 0 |
| GGGUAG CGUAGAUCU | 2 | 0 |
| GUGUAG CAUAGAGAU | 2 | 0 |
| GUGUAC GGUAGAUAU | 1 | 2 |
| GAGUAG CGUAGAUAC | 1 | 2 |
| GAGGAG CGUUGAUAC | 1 | not found |
| GUGUAA UGUUGAUAU | 1 | 2 |
| GUGGAG CGUUGAUAU | 1 | 1 |
| GUGUAG UGGAGAUAU | 1 | 2 |
| GUGUAG CGUACAGAU | 1 | 0 |
| GUUGUAG CGUAGAGAU | 1 | 1 |
| GUGAAG CGUAGAUAU | 1 | 1 |
| GUGAAG UGUAGAUAU | 1 | 2 |
| GUGUAG CCGUAGAGAU | 1 | 1 |

Table 5.1: Motifs with occurrences in the multiple alignment and results from searching algorithm

```
            )))))))..))))))))).(.(((.(((....((((.(((...(((((([****[((...
seq1        CACCUGAUACUGCUGGACUUGAGGGCAGGAGAGGAGAGUGGAAUUCCCGGUGUAGCGGUG
seq2        CAUUCGAAACUGGCAGGCUUGAGUCUUGUAGAGGGGGGUAGAAUUCCAGGUGUAGCGGUG
            )))))))..))))))))).(.(((.(((....((((.(((...(((((([****[((...

            ...))]*******]))))))...))).(((.((....)).)))))))...)))..))).)
seq1        AAAUGCGUAGAUAUCGGGAGGAACACCAGUGGCGAAGGCGGCUCUCUGGACUGAACCUGA
seq2        AAAUGCGUAGAGAUCUGGAGGAAUACCGGUGGCGAAGGCGGCCCCCUGGACAAAGACUGA
            ...))]*******]))))))...))).(((.((....)).)))))))...)))..))).)
```

Table 5.2: Output of *LocARNA* with motif detection

# Chapter 6

# Conclusion

The function of RNA mainly depends on its 3D structure. However, it is hard to observe and computationally expensive to predict. Sequence-structure alignment algorithms align RNAs on the primary and secondary structure, which do not totally reflect the function of the RNA. Observation of known 3D structures has revealed repeated structural motifs, which are arrays of non-Watson–Crick base pairs. These motifs can be described on sequence level, and can be efficiently detected using string searching algorithms. These motifs can give a strong guidance for alignment software. Moreover, secondary structure prediction is erroneous and thus supporting information due to 3D motifs help correcting the structure.

In construct a motif alignment algorithm, all isosteric motifs were generated using a recursive algorithm. Two string searching algorithms were implemented in order to search the list of motifs inside the RNA sequences to be aligned. One algorithm is an exact searching algorithm while the other allows errors in the pattern or text. The search for motifs was merged with *LocARNA*, an already existing sequence-structure algorithm.

A list of isosteric motifs was produced for the aim of detecting the 3D motifs on the structure level. It resulted in a total of 1536 isosteric motifs. The improved *LocARNA* searches successfully for structural motifs inside the RNA sequences, and matches them with a higher score than normal sequence or structure matching. It includes the option of exact matching of patterns or with variable number of errors.

Open problems for future research include increasing the number of motifs. The list of motifs used in this study consists of the motifs observed in the kink-turns of ribosomal RNA. However, there exist other motifs observed in C-loops, hairpin loops, internal loops and junction loops [2, 6]. The addition of other types of motifs requires changes to be done to the algorithm for aligning the motifs and to the scoring function. Some types of motifs can be composed of more than two parts (two ranges), their detection will require changes in the searching algorithm and the motif list construction algorithm.

The modified *LocARNA* has been only tested on the 16S ribosomal RNA family. Extensive tests on biological data are required to prove the effectiveness of the algorithm on different RNA families. Finally, the motif detection can be integrated in a multiple alignment tool based on pairwise alignments.

# Bibliography

[1] Rolf Backofen and Sebastian Will. Local sequence-structure motifs in RNA. *Journal of Bioinformatics and Computational Biology*, 2(4):681–698, 2004.

[2] Steven E. Brenner Donna K. Hendrix and Stephen R. Holbrook. Rna structural motifs: building blocks of a modular biomolecule. *Quarterly Reviews of Biophysics*, 38(3):221–243, 2006.

[3] http://www lbit.iro.umontreal.ca/mcfold/logo.mcfold.png.

[4] N. B. Leontis and E. Westhof. Geometric nomenclature and classification of rna base pairs. *RNA*, 7(4):499–512, 2001.

[5] Neocles B. Leontis, Jesse Stombaugh, and Eric Westhof. The non-watson-crick base pairs and their associated isostericity matrices. *Nucleil Acids Research*, 30(16):3497–3531, 2002.

[6] Aurelie Lescoute, Neocles B. Leontis, Christian Massire, and Eric Westhof. Recurrent structural rna motifs, isostericity matrices and sequence alignments. *Nucleic Acids Research*, 33(8):2395–409, 2005.

[7] J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for rna secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990.

[8] David Sankoff. Simultaneous solution of the rna folding, alignment and protosequence problems. *SIAM Journal on Applied Mathematics*, 45(5):810–825, October 1985.

[9] Sven Siebert and Rolf Backofen. Marna: multiple alignment and consensus structure prediction of rnas based on sequence structure comparisons. *Bioinformatics*, 21(16):3352–3359, 2005.

[10] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[11] Gabriele Varani and William H. McClain. The g.u wobble base pair: A fundamental building block of rna structure crucial to rna function in diverse biological systems. *EMBO reports*, 1(1):18–23, 2000.

[12] Z. Vokacova, J. Sponer, and V. Sychrovsky. Theoretical nmr study of watson-crick/sugar edge rna base pair family.

[13] Eric Westhof and Pascal Auffinger. *RNA Tertiary Structure*, chapter ? John Wiley & Sons Ltd, Chichester, 2000.

[14] Sebastian Will, Kristin Reiche, Ivo L. Hofacker, Peter F. Stadler, and Rolf Backofen. Inferring non-coding rna families and classes by means of genome-scale structure-based clustering. *PLOS Computational Biology*, 3(4):e65, 2007.

[15] Sun Wu and Udi Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, October 1992.