Algorithms for finding RNA sequence-structure motifs

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
am Fachbereich Mathematik und Informatik
der Freien Universität Berlin

vorgelegt von
Jörg Winkler

Berlin, 2022

1st supervisor	Prof. Dr. Knut Reinert
2nd supervisor	Prof. Dr. Annalisa Marsico
Date of disputation	08.09.2023

The function of non-coding RNA sequences is largely determined by their spatial conformation. This is the secondary structure of the molecule, which is formed by Watson–Crick interactions between nucleotides. Hence, modern RNA alignment algorithms routinely take structural information into account. Essential tasks for discovering yet unknown RNA families and inferring their possible functions are the structural alignment of RNAs and the subsequent search of the derived structural motifs. These tasks demand a lot of computational resources, especially for aligning many long sequences, and it therefore requires efficient algorithms that utilize modern hardware when available. A subset of the secondary structures contains pseudoknots, which are overlapping interactions that add additional complexity to the analysis and are often ignored in available software.

In this thesis I present LaRA 2 and MaRs, two SeqAn-based software tools that implement algorithms for finding sequence-structure motifs in genomic sequences. In contrast to other programs my tools can handle arbitrary pseudoknots. They use multithreading for parallel execution and are implemented in modern C++ code for maximal longevity and performance.

LaRA 2 is significantly faster than comparable software for accurate pairwise and multiple alignments of structured RNA sequences. It uses a new heuristic for computing a lower boundary to the solution and employs vectorization techniques for speeding up the time-critical parts of the algorithm.

MaRs can be applied in a workflow right after LaRA 2 and derives sequence-structure motifs from the structural alignments. The motifs are descriptors of the RNA sequences' properties and drive the search for homologs in genomic sequences. MaRs employs a bi-directional index on the genomic sequences and an optimized multithreaded search strategy for finding the matches really fast. The use of a thread pool, effective pruning strategies, and a low memory footprint ensure that MaRs is capable of processing extremely large data sets.

Contents

List of Figures

List of Tables

List of Listings

Part I

Background

1 Ribonucleic acid

RNA, short for ribonucleic acid, is a versatile biopolymer that is typically single-stranded. It has a backbone composed of ribonucleotides, which are linked by phosphodiester bonds. A ribonucleotide consists of a ribose sugar, a phosphate group, and one of the four nucleobases adenine (A), cytosine (C), guanine (G), or uracil (U). We use the characters A, C, G, and U to describe the sequence of nucleotides in an RNA molecule, which is also called primary structure [Picardi, 2015, Parker et al., 2016].

Similar to DNA (deoxyribonucleic acid), the nucleobases are able to build hydrogen bonds with their complement: The complement of cytosine is guanine, and the complement of adenine is uracil. However, as RNA consists usually of a single strand, it extensively builds intramolecular base pairings between complementary parts. The pattern of these pairings is denoted as secondary structure and is deeply discussed in section 1.3. Before that, let us look at different types of RNA (section 1.1) with a focus on long non-coding RNA (section 1.2).

1.1 Types of RNA

According to the central dogma of molecular biology, the genetic information is transcribed from DNA to (coding) RNA, which in turn gets translated at the ribosomes into proteins [Crick, 1958, Crick, 1970]. It came as a surprise to researchers, when it was discovered that the majority of RNA in a eukaryotic cell did not code for proteins. In fact, in mammalian cells the content of messenger RNA (**mRNA**), which is the coding RNA, is only 3–7 % of the mass of total RNA [Palazzo and Lee, 2015]. mRNA is either monocistronic, if it translates a single protein, which is usually the case for eukaryotic mRNA, or polycistronic, if it can translate multiple proteins [Kozak, 1983]. Since it became clear that the developmental and physiological complexity of humans cannot be explained by protein-coding genes alone, the research on non-coding RNA has drastically expanded [Wilusz et al., 2009].

Non-coding RNA (ncRNA) can be further categorized into housekeeping and regulatory ncRNA [Ponting et al., 2009]. Housekeeping ncRNAs are expressed constitutively, and for many cellular processes they work as key regulatory molecules [Losko et al., 2016]. As figure 1.1 shows, there are ribosomal (rRNA), transfer (tRNA), small nuclear (snRNA) and small nucleolar (snoRNA) RNAs in this group:

rRNA Ribosomal RNA is with 80–90 % the vast majority of RNA mass in a cell [Palazzo and Lee, 2015]. As the name suggests, this type of ncRNA is the main component of ribosomes, where protein synthesis takes place.

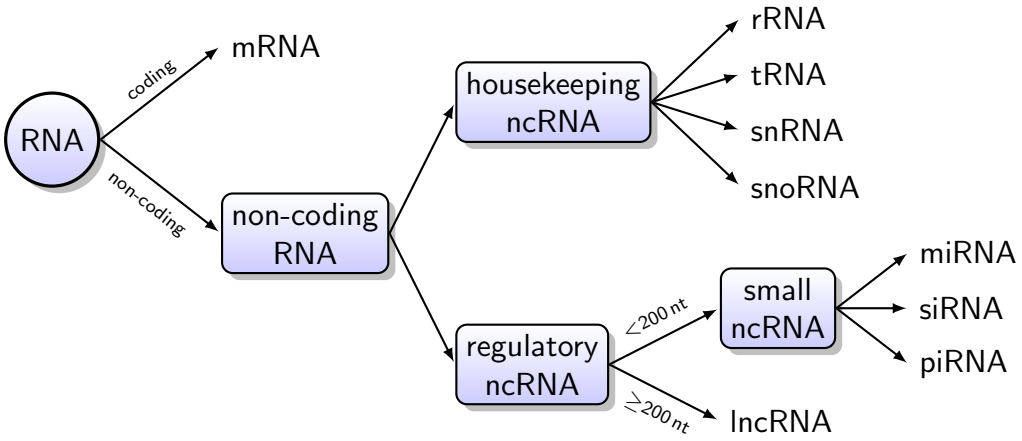


Figure 1.1: **A systematic scheme of RNA types.** The main distinction is made between the messenger RNA that contains the code for building proteins and the group of non-coding RNAs. This group is further divided into housekeeping ncRNAs, which are necessary for elementary functions of the cell, and regulatory ncRNAs, which play important roles in gene regulation, and are still subject to ongoing research.

tRNA Transfer RNA represents 10–15 % of a cell's RNA. It is rather small and even though there is less tRNA by mass than rRNA, the number of tRNA molecules is about 10 times higher [Palazzo and Lee, 2015]. Its secondary structure looks like a cloverleaf and while it accepts an amino acid on the one side, it has the anti-codon on the opposite side. The anti-codon are three bases that are complement to the code for the respective amino acid.

snRNA Small nuclear RNA (0.02–0.3 %) has about 150 nucleotides and is located in the chromatin. It performs post-transcriptional modifications of mRNA and builds the spliceosome for intron removal and maturation of mRNA.

snoRNA Small nucleolar RNA (0.04–0.2 %) has 60-300 nucleotides and is located in the chromatin. It performs post-transcriptional modifications and maturation of rRNA, snRNA and other cellular ncRNAs.

The group of regulatory ncRNAs has a large variety of functions like gene expression control and enzymatic catalysis during splicing and translation. Regulatory ncRNAs are divided into long and short ncRNA, where long ncRNA (**lncRNA**) has at least 200 nucleotides. Long ncRNA can be categorized into groups by location, sequence, morphology, structure and function features, and therefore represents the broadest class of ncRNAs [Losko et al., 2016]. As it also has the highest importance in this thesis, the next section 1.2 is solely dedicated to lncRNA. For short ncRNA with less than 200 nucleotides the most important types are micro RNA (miRNA), small interfering RNA (siRNA), and PIWI interacting RNA (piRNA):

miRNA Micro RNA is a very short (about 20–23 nucleotides), highly conserved ncRNA. It plays an essential role in post-transcriptional regulation of gene expression, and can silence mRNA molecules very specifically, e.g. by base-pairing with complementary sequences [Bartel, 2009, Laganà et al., 2015].

siRNA Small interfering RNA is a double-stranded molecule with a length of 20–25 base pairs. It silences exogenous nucleic acids and undesired transcripts and therefore maintains the genome integrity and is involved in the cell defence [Laganà et al., 2015].

piRNA PIWI interacting RNA has a length of 21–35 nucleotides. It regulates gene expression, silences transposable elements, and inhibits viral infection [Monga and Banerjee, 2019]. PIWI is a regulatory protein with RNA binding site that is responsible for stem cell and germ cell differentiation [Cox et al., 2000].

1.2 Biological relevance of long non-coding RNA

Long non-coding RNAs (lncRNAs) exhibit a surprisingly wide range of sizes, shapes and functions in comparison to other RNAs. These features provide lncRNAs with huge functional potential, as they have roles in all different aspects of gene expression. There is currently much research carried out in order to study their functions, and we still have little knowledge on this class of RNA. Experiments for their analysis are very challenging, as their various functions depend on many aspects, like subcellular localization, attraction to interaction partners, and dynamic changes in local cell environments [Yao et al., 2019]. For validating the role of lncRNA, researchers often conduct gain- or loss-of-function experiments, but it is very difficult to determine which cellular process can be probed to yield an observable phenotype. In order to keep pace with the fast progress in lncRNA discovery, more efficient analyses and high-throughput approaches are needed [Sun and Kraus, 2015].

Transcription is not limited to protein-coding regions, and it is likely that over 90 % of the human genome is transcribed [ENCODE Project Consortium et al., 2007]. Most of the transcribed non-coding sequence is associated with lncRNA. However, Ponting et al., 2009 hypothesize that a large proportion of proposed lncRNAs may instead be artefacts of either experiment or computation, and represent e.g. fragments of unprocessed pre-mRNAs. Nevertheless, a large amount of lncRNAs have been studied that are regulated during development, exhibit a cell type specific expression, localize to specific subcellular compartments, and are associated with human diseases [Wilusz et al., 2009]. This shows that lncRNAs are, besides proteins, a cell's key regulatory molecules, as shown in figure 1.2.

LncRNAs serve as an organizational framework of subcellular structures [Wilusz et al., 2009]. For instance, they act as a molecular scaffold to recruit and combine with multiple regulatory proteins. Some attach to both DNA and protein, and add methyl groups to the DNA, which can be interpreted as tagging parts of DNA as active or inactive. Furthermore, they mediate chemical modifications to histone

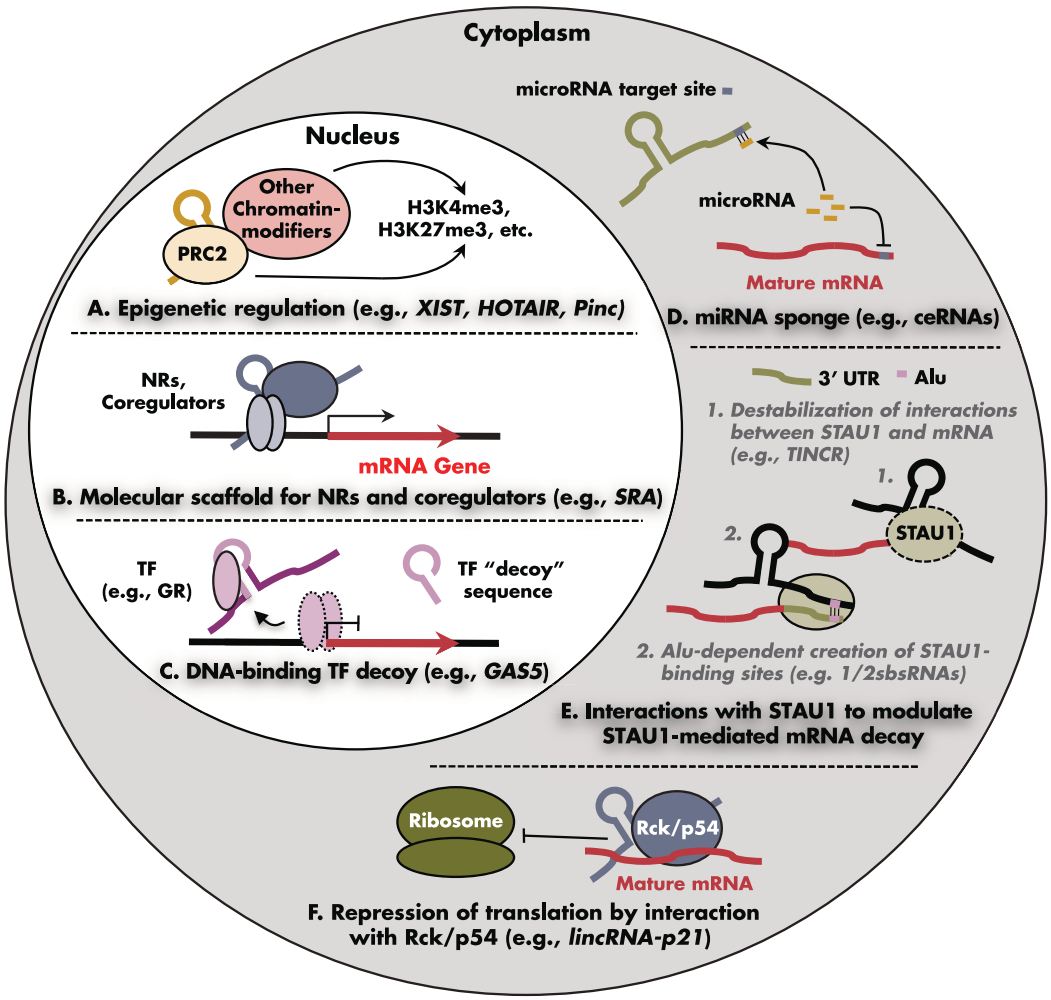


Figure 1.2: **Different regulatory functions of lncRNA** in the nucleus and cytoplasm. The image is taken from Sun and Kraus, 2015. The nuclear functions are interactions with chromatin modifiers (A) and transcription factors (B) to alter epigenetic modifications and gene regulatory activities, as well as inhibiting the activity of DNA-binding transcription factors (C). In the cytoplasm, lncRNAs act as a sponge to miRNA to reduce their targeting of mRNA (D), as a regulator for mRNA stability through binding with the STAU1 protein (E), and as translation inhibitor (F).

proteins, which are associated to DNA and form chromatin fibres. The interaction of DNA methylation patterns, histone modifications, and chromatin structure is the central component of epigenetics, a field of study that aims to explain heritable phenotypical changes that are not caused by DNA sequence alterations. In addition, lncRNAs regulate the activity and location of proteins, and through allosteric effects they are able to interact with different ligand proteins. Even the act of ncRNA transcription alone can be sufficient to affect the expression of nearby genes positively or negatively [Wilusz et al., 2009].

It is now well-established that lncRNA molecules introduce an additional layer in genetic information processing. They play a significant, active role in cell and developmental biology and carry out many tasks that were previously attributed exclusively to proteins. However, only a small fraction of lncRNA families have been identified so far and many more can still be discovered [Mattick, 2005]. Structural RNA elements are also involved in the control of virus replication [Viehweger et al., 2019], transcription and translation, indicating that the usage of the RNA structure features will be exploited in the near future for designing novel antiviral strategies [Lim and Brown, 2017].

1.3 Structure of ncRNA

Comparing functionally related ncRNA molecules requires more than sequence information, because their function is primarily determined by their secondary structure, which is often better conserved than the primary sequence. Hence, sequence-structure alignments reward the conservation of structural interactions of the ncRNA molecules, which is a key property for many applications, e.g. finding homologous structures of known ncRNA families [Kalvari et al., 2018], phylogenetic fingerprinting as conducted for example for the ITS2 database [Wolf et al., 2005], or the computation of a consensus structure of a set of related RNA molecules [Hofacker et al., 2004, Torarinsson et al., 2007, Bauer et al., 2007, Will et al., 2007, Xu and Mathews, 2011, Tabei et al., 2008, Wei et al., 2011, Meyer and Miklós, 2007, Tan et al., 2017].

Owing to the importance of ncRNA molecules, there has been a steady stream of developments for analysing the molecules computationally. Specific rules govern RNA structure formation, therefore structured RNAs provide clear patterns of selection with base pairing patterns directly reflecting structural conservation [Rivas et al., 2017]. In other words, two nucleotides that form a base pair may be changed by mutations but preserve the propensity to form a valid base pair through compensatory mutations. Having a good model of an RNA structure (or a secondary structure as proxy of the 3D structure) is therefore crucial to elucidate its function [Gutell et al., 1992].

Different algorithms for computing the best secondary structures from an RNA sequence are discussed in section 2.1. However, it is important to note that there is generally not a single conformation of an RNA structure in a living cell, as it is very dynamic and can be dependent on different reaction conditions, like temperature, somatic variation, metabolite concentration, ATP depletion, and knockdown of interacting proteins [Qian et al., 2019].

For the analysis of secondary structure it is helpful to decompose the whole structure into different parts, based on the presence or absence of hydrogen bonds within the molecule. Hydrogen bonds are formed between the canonical Watson-Crick pairs, which are the stable complementary `CG` and `AU` base pairs, as well as between Wobble pairs, which denote weaker `GU` pairings. The more stable an interaction is, the more often it occurs. Thus, the frequency of different base pairs across all types

	A	C	G	U
A	2.22%	1.81%	5.66%	22.94%
C		0.36%	57.21%	0.59%
G			0.74%	7.40%
U				1.08%

Table 1.1: **The relative base pair composition in RNA helical structures.** Canonical and Wobble pairs are highlighted. The numbers are based on the structures present in the Nucleic Acid Database [Berman et al., 1992] and are computed from absolute values published in Olson et al., 2009.

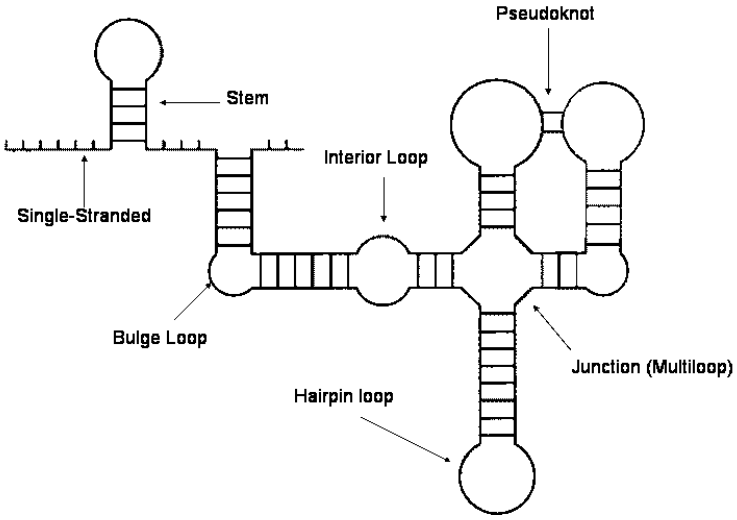


Figure 1.3: **Secondary structure elements.** A secondary structure consists of a collection of the following elements: stem, hairpin, multiloop, interior loop, bulge loop, pseudoknot, and single-stranded regions. The image is taken from the rPredictorDB user documentation[1].

of ncRNAs, which is shown in table 1.1, gives a good picture of their likelihood and chemical stability.

Based on the locations of base pairs, the following structure components are considered. Figure 1.3 represents them graphically.

Single-stranded region This is a part of a sequence that is not involved in any base pairings. It is also called *Loop*.

Stem A stem consists of a sequence of consecutive base pairings. It requires two parts of the sequence to have reverse complementary nucleobases.

Hairpin loop It is a single-stranded region, which links the two ends of a stem.

[1] http://rpredictor.ms.mff.cuni.cz/documentation/_images/secondary-structures.png (01.06.2021)

Interior loop An interior loop consists of two single-stranded regions, which are located on both strands between two stems.

Bulge loop This is a single-stranded region, which is located on one strand between two stems. On the opposite strand the two stems are adjacent to each other. It can be also considered as a special interior loop, where one of the loops has length zero.

Multiloop A junction that possibly contains single-stranded regions and is adjacent to multiple stems.

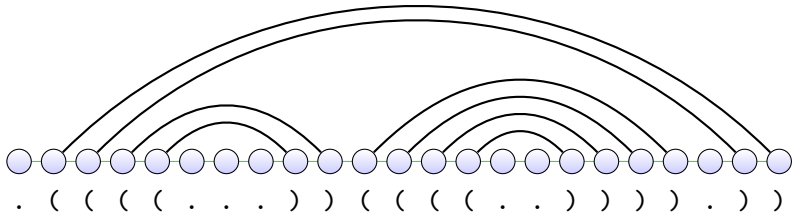
Pseudoknot Crossing interactions of loop regions. Typical pseudoknots are between two hairpin loops (so-called *Kissing Hairpins*, as shown in figure 1.3), but also other types are possible, e.g. hairpin loop with bulge loop. As pseudoknots play an important role in this thesis, the next section is devoted to them.

As a compact representation of a secondary structure, the dot-bracket notation has been established by Hofacker et al., 1994. In its basic form, it uses the dot character to denote an unpaired site and matching parentheses to denote a base pair. For instance, a stem of length 3 with a hairpin loop of length 4 is written as `(((....)))`. A dot-bracket string always has the same length as its corresponding sequence and with only a single bracket type it requires structures to be non-crossing [Hofacker et al., 1994]. For pseudoknots, the dot-bracket notation has been extended by additional bracket types, like curly, squared and angle brackets (`{}[]<>`), as well as matching pairs of uppercase and lowercase letters (`Aa Bb` etc.). This is the so-called WUSS (Washington University Secondary Structure) notation.

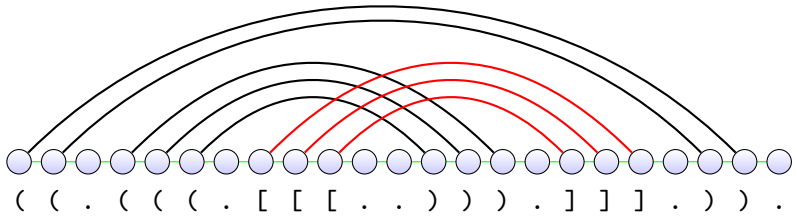
1.4 Pseudoknots

About 12% of known RNA structures contain pseudoknots [Danaee et al., 2018], which are crossing interactions of loop regions. In figure 1.4, the difference between secondary structures without and with pseudoknots is visualized. Within pseudoknotted structures the base pairing is not well nested, i.e. for two base paired positions (i, j) and (h, k) either $i < h < j < k$ or $h < i < k < j$ holds. In other words, base pairs overlap each other with respect to their sequence position. Pseudoknot base pairs are annotated as the minimal set that results in a pseudoknot-free structure when removed [Danaee et al., 2018]. In figure 1.4b the respective base pairs are marked in red.

The terms *page number* or *book thickness* of a pseudoknotted structure [Haslinger and Stadler, 1999] are used to denote the complexity of a pseudoknot. If we consider a *page* as a set of nested (i.e. pseudoknot-free) interactions, then the page number is defined as the minimal number of pages needed to describe the structure [Haslinger and Stadler, 1999]. From this follows that a pseudoknot-free structure always has page number 1, while a structure with a pseudoknot has at least page number 2. In



(a) In a nested structure the interactions do not cross each other.



(b) In a structure with a pseudoknot some interactions are crossing.

Figure 1.4: **Pseudoknots have crossing (non-nested) interactions.** The comparison of two linearly drawn structures without and with a pseudoknot (marked in red) is shown. The blue circles represent the nucleobases, and the green line marks the backbone of the RNA molecule. Below the nucleobases there is the dot-bracket notation of the structure: While for a nested structure one type of parentheses is sufficient, we need to use at least two types to represent a pseudoknot.

terms of the extended dot-bracket notation introduced in the previous section, the page number can be interpreted as the minimal number of bracket types that are required in order to represent all the base pairs of a structure.

Pseudoknots are difficult to predict with standard methods that use dynamic programming or stochastic context-free grammars that rely on the nested property [Jabbari et al., 2018]. In fact, the majority of today's software for structure prediction and alignment does not recognize pseudoknots, and the programs that do support them are more complex and are therefore more limited regarding the input size [Rivas and Eddy, 1999, Dirks and Pierce, 2004, Möhl et al., 2010].

From a biological viewpoint, pseudoknots must not be neglected, as they play various important roles. For instance, they can be catalytically active or induce ribosomes to slip into alternative reading frames, which results in altered gene expression [Staple and Butcher, 2005]. This gives rise to the need for algorithms that can actually predict and analyse pseudoknots in RNA structure. The programs `LaRA 2` and `MaRs`, which are discussed later on in this thesis, are both able to analyse pseudoknotted RNA.

2 Computational RNA analysis

In the light of the biological importance of RNA that we have discussed in the previous chapter, let us take a look at the computational analysis of RNA molecules. In the last decades, much progress has been made in the computation of the most likely structures of an RNA molecule, as well as in the comparison of these structures.

This chapter is divided into three sections. Section 2.1 focuses on secondary structure prediction and demonstrates how structure predictions are obtained from one or more sequences. Having several related sequences with corresponding structures available, the best way of extracting information is a sequence-structure alignment, which we discuss in section 2.2. We look at methods for structure alignments, and observe that we can identify conserved structural motifs from these alignments. The positions of these structural motifs are likely the functional sites of the respective RNA class. Therefore, in the final section 2.3, we see how RNA can be classified based on conserved structure or sequence features.

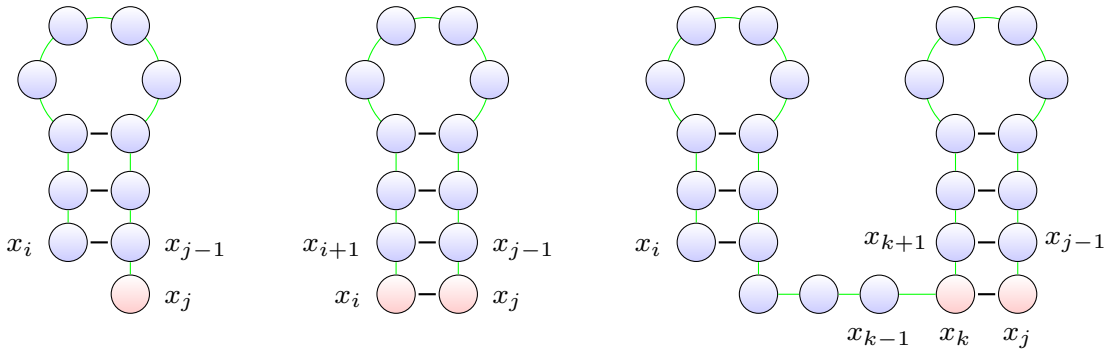
2.1 Prediction of secondary structure

A secondary structure is a simplification of the complex, three-dimensional folding of an RNA molecule, as we have seen in section 1.3. Besides the backbone, it emphasizes the hydrogen bonds between the nucleobases within the RNA molecule. With the knowledge which bases pair with each other, we can computationally predict secondary structures. Note that each nucleobase can only interact with one partner and that the interactions are undirected.

2.1.1 Base pair maximization

The easiest approach is to maximize the number of canonical base pairs, as it is the purpose of the Nussinov algorithm. It is a recursive algorithm that employs dynamic programming by calculating the best structure for small subsequences and extending them step by step. This algorithm cannot detect pseudoknots and the stability of GC pairs is considered equal to AU pairs [Nussinov and Jacobson, 1980].

The input is a sequence $x = (x_1, x_2, \ldots, x_L)$ of length L. We compute a matrix $\mathcal{N}$ of size $L \times L$ that contains at $\mathcal{N}(i, j)$ the maximal number of base pairs that can be formed for the subsequence $(x_i \ldots x_j)$. Thus, the maximal number of base pairs in the whole sequence is the value $\mathcal{N}(1, L)$.



(a) Extension of a substructure with a base x_j.

(b) Extension of a substructure with a base pair x_i, x_j.

(c) Combination of two substructures with a closing base pair x_k, x_j.

Figure 2.1: **The case distinction in Nussinov's algorithm.** The blue substructures are extended by the red nucleobases. The indices of relevant nucleobases are shown according to equation (2.2). The green line indicates the backbone of the exemplary RNA molecule.

Because of the biological constraint that a nucleobase does not interact with itself nor its neighbour, we initialize the values for very short subsequences with zero:

$$\begin{aligned} \text{for } 1 \leq i \leq L: \qquad & \mathcal{N}(i,i) = 0 \\ & \mathcal{N}(i,i-1) = 0 \end{aligned} \tag{2.1}$$

We define the notation $x_i \bowtie x_j$, which denotes the condition that x_i and x_j complement each other. The remaining matrix entries can be calculated with the recurrence:

$$\text{for } 1 \leq i < j \leq L: \qquad \mathcal{N}(i,j) = \max \begin{cases} \mathcal{N}(i,j-1) \\ \max\limits_{\substack{i \leq k < j \\ x_k \bowtie x_j}} \mathcal{N}(i,k-1) + \mathcal{N}(k+1,j-1) + 1 \end{cases} \tag{2.2}$$

The first case adds an unpaired base x_j to the optimal substructure of $(x_i \dots x_{j-1})$ without increasing the score, as shown in figure 2.1a. In the second case, for $k = i$, we increment the score for adding a complementary base pair x_i, x_j to the optimal substructure of $(x_{i+1} \dots x_{j-1})$, as the value evaluates to $0 + \mathcal{N}(i+1, j-1) + 1$. This case is demonstrated in figure 2.1b. Furthermore, adding an unpaired base x_i to the optimal substructure of $(x_{i+1} \dots x_j)$ does not increase the score, because the condition $x_i \bowtie x_j$ is false. For $k > i$ we introduce a bifurcation, as shown in figure 2.1c: The two optimal substructures of $(x_i \dots x_{k-1})$ and $(x_{k+1} \dots x_{j-1})$ are combined, while adding a final matching base pair x_k, x_j to the second substructure, which implies a score increment.

$\mathcal{N}(i,j)$ is filled with sections of increasing length $j - i$, so the entries $\mathcal{N}(i, j-1)$, $\mathcal{N}(i, k-1)$, and $\mathcal{N}(k+1, j-1)$ have been already computed before and can be

read from the matrix. Therefore, the algorithm runs in $\mathcal{O}(L^3)$ time and $\mathcal{O}(L^2)$ space. The maximal number of base pairs for the whole sequence x is the value in cell $\mathcal{N}(1, L)$. Starting from this cell, we can trace back the recursion cases that yielded the maximum for the respective cell, in order to retrieve an optimal secondary structure. The trace-back is computed in $\mathcal{O}(L^2)$ time [Nussinov and Jacobson, 1980].

However, the algorithm comes with some limitations: Crossing structures cannot be predicted, so the result never contains pseudoknots. Furthermore, the base pair maximization does not differentiate the structures well enough, so that multiple, quite different optimal structures may exist with the same base pair count. Therefore, it is not enough to report only one structure. This can be overcome with extensions of the algorithm or for instance with the algorithm by Wuchty et al., 1999. Nevertheless, the structures are biologically not very relevant, because important aspects are neglected, like the profit of base pair stacking, different loop sizes and the properties of multi-loops. The Zuker algorithm, which is introduced in the following subsection, improves this by defining energy terms for these structure properties.

2.1.2 Free energy minimization

Instead of maximizing the number of base pair interactions, we use energy terms to obtain a more realistic picture of the stability of an RNA structure. The free energy of a structure is a measure for its thermodynamic instability, which means that a structure with high free energy is not stable and likely to change, while a structure with low free energy is rather stable and in theory more likely to occur.

The algorithm by Zuker and Stiegler, 1981 computes for a given RNA sequence the optimal secondary structure, considering the free energy contributions of different types of structure components. For each hairpin, stem, interior loop, and multiloop (compare figure 1.3) an individual free energy value is obtained based on their properties: while unpaired sites in a loop increase the free energy amount, canonical base pairs have a negative contribution. A bulge loop is considered as an interior loop with one empty loop, and pseudoknots are not detected. The free energy of the whole RNA structure is the sum of the energies of its components.

In order to find the minimum free energy (MFE) structure, the algorithm has to visit each possible conformation and compare their energies. This is performed efficiently in the style of the Nussinov algorithm: a recursion starts from the shortest interactions and extends these substructures until the whole sequence is covered. The algorithm runs in $\mathcal{O}(L^2)$ space and $\mathcal{O}(L^4)$ time to fold a sequence of length L.

The run time complexity is caused by the interior loops, where for each base pair (x_i, x_j) we are looking for another base pair (x_h, x_k) with $h < i < j < k$ that closes the loop. This can be overcome by a heuristic that limits the loop size to a constant amount, but it may compute a suboptimal structure. Another approach by Lyngsø et al., 1999 uses an optimization for the interior loops that reduces the time complexity to $\mathcal{O}(L^3)$ by utilizing currently used energy rules.

The obtained MFE structure gives us a much more realistic picture of the interactions that an RNA sequence builds, compared to the base pair maximization.

However, a single structure is often not enough for reliable RNA comparisons, as we see in the following subsection.

2.1.3 Base pair distribution

In section 1.3, we discussed that there is not a single conformation of an RNA structure, but rather a large variety of structures, of which we try to find the most likely ones. Therefore, it is more meaningful to compute individual probabilities of base pairings, which represent the whole ensemble of structures, than a single minimum free energy (MFE) structure.

Let Ψ be the set of all possible structural conformations of an RNA sequence. In equilibrium, the likelihood of the structures are determined by the Boltzmann distribution, so that the probability p_ψ of a particular structure $\psi \in \Psi$ can be calculated as follows:

$$p_\psi = \frac{1}{Z} \mathrm{e}^{\frac{-E(\psi)}{RT}} \qquad \text{with} \qquad Z = \sum_{\psi \in \Psi} \mathrm{e}^{\frac{-E(\psi)}{RT}}$$

The parameter $E(\psi)$ is the free energy of conformation ψ, T is the temperature, and R the universal gas constant. The normalization denominator is the canonical partition function Z, which is the sum of the probabilities of all accessible conformations [Pałkowski and Bielecki, 2019].

Knowing the probability of a particular structure enables us to calculate base pairing probabilities. Let $\Psi_{ij} \subseteq \Psi$ be the set of all possible structural conformations of sequence x, in which x_i and x_j form a base pair. Then the probability $\mathcal{P}(i, j)$ that x_i and x_j form a base pair can be calculated as the sum of the probabilities of all these structure conformations:

$$\mathcal{P}(i, j) = \sum_{\psi \in \Psi_{ij}} p_\psi$$

The base pair probabilities can be well visualized in a so-called dot plot, as shown in figure 2.2. It shows a matrix labelled with the sequence in both dimensions, and there are black dots in the matrix cells, whose size represents the probability of the base pairing between the respective nucleobases.

McCaskill, 1990 introduced an algorithm to efficiently compute the partition function in $\mathcal{O}(L^2)$ space and $\mathcal{O}(L^3)$ time for a sequence of length L. It follows the dynamic programming scheme of the algorithm by Nussinov and Jacobson, 1980, i.e. we start with the computation of the partition functions of small subsequences, store the intermediate results in a matrix, and extend until the whole sequence is covered. It employs the energy model by Zuker and Stiegler, 1981 with the optimization to $\mathcal{O}(L^3)$ complexity by Lyngsø et al., 1999.

As we have seen above, the base pair probabilities can be obtained by adding up the terms of the partition function, whose structure contains the respective base pair. Although the considered structures in the partition function cannot contain

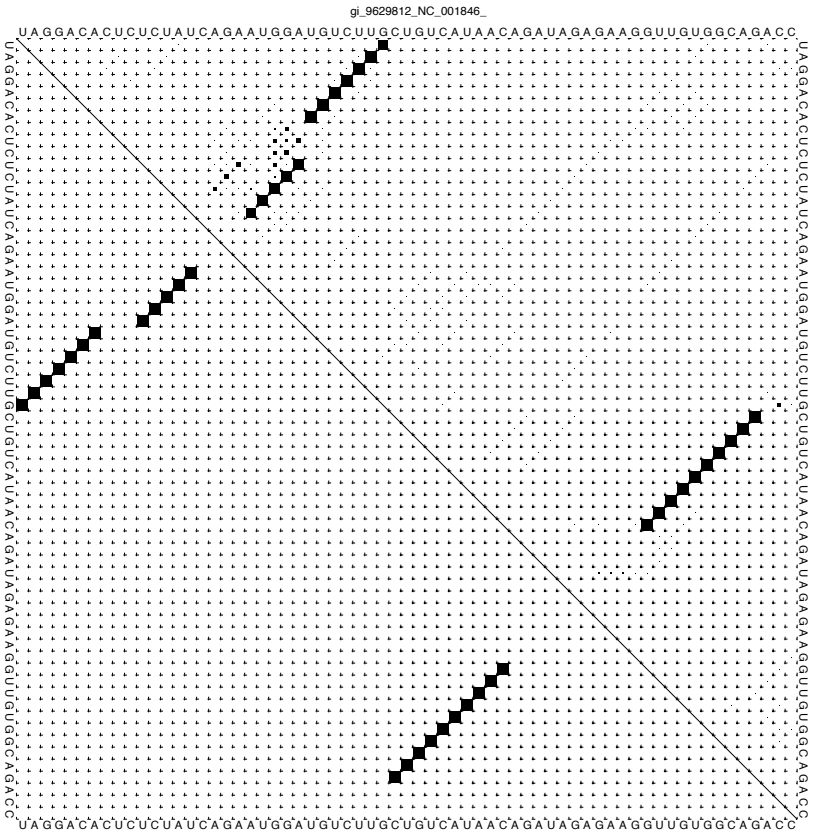


Figure 2.2: **A dot plot represents base pair probabilities.** A subsequence of a murine hepatitis virus (accession NC_001846) is displayed in both horizontal and vertical directions and the plot shows for each pair of nucleobases the probability that this pair forms an intramolecular interaction. In the upper triangular matrix the size of the dots corresponds to the probabilities: The larger a dot is drawn, the higher is the probability it represents. The lower triangular matrix displays only the optimal structure, i.e. there is at most one dot per column or row.

pseudoknots through the application of dynamic programming [McCaskill, 1990], the resulting base pair probabilities may still point out the interactions that are likely involved in a pseudoknot. This would not be possible if only one optimal structure was computed, as the algorithm needs to decide for the strongest of crossing interactions and omit the other.

The algorithm is implemented in the `RNAfold` tool [Lorenz et al., 2011], which is widely used. Recently, parallel and cache-efficient code of the algorithm for the TRACO and PLUTO compilers have been developed [Pałkowski and Bielecki, 2019,Zhao and Sahni, 2020]. Furthermore, extensions of the algorithm were published that include pseudoknots, but they are not feasible for the analysis of long sequences: The algorithm by Rivas and Eddy, 1999 requires $\mathcal{O}(L^6)$ time and $\mathcal{O}(L^4)$ space, and the algorithm by Dirks and Pierce, 2004 runs in $\mathcal{O}(L^5)$ time and $\mathcal{O}(L^4)$ space.

2.2 Sequence-structure alignment

The computation of biologically meaningful alignments is challenging for RNA sequences whose nucleobase identity is under 60% [Capriotti and Marti-Renom, 2010]. As in RNA molecules the secondary structures are evolutionary conserved, covariation often becomes the strongest available signal. Standard sequence aligners like `ClustalW` [Thompson et al., 1994], `MAFFT` [Katoh et al., 2002], or `T-Coffee` [Notredame et al., 2000] assume site independence and cannot take this information into account [Chatzou et al., 2016].

In a set of homologous RNA sequences, we want to find conserved elements, so-called motifs, that are likely relevant for the RNA's function. These motifs can be extracted from a more specialized alignment that takes both the sequence and the structure into account. The existing methods for RNA alignment can be divided into three categories: (1) first align then fold, (2) first fold then align, and (3) sequence-structure alignment.

The first category computes a sequence alignment of the RNA sequences, and then tries to find the best MFE structure that represents the alignment. Existing software tools for this approach are e.g. `SeqAn` [Reinert et al., 2017], `MAFFT` [Katoh et al., 2002], and `ClustalW` [Thompson et al., 1994] for the alignment, and `RNAalifold` [Lorenz et al., 2011], `RNAalishape` [Janssen and Giegerich, 2015], and `Pfold` [Knudsen and Hein, 2003] for the secondary structure. One problem with this approach is that the sequence alignment must be very good to form a basis for structure prediction, i.e. it must correctly align the biologically correspondent nucleobases of the sequences. This is surprisingly often not the case, as a recent protocol by Warnow, 2021 shows. The other problem is that functional RNA structures often reside in non-conserved parts of the sequence [Pervouchine, 2018]. This is unfortunate for this approach, as the regions of the highest uncertainty in the alignment are the most important for the structure prediction.

The second category finds a secondary structure for each individual sequence, either computationally as discussed in the previous section or experimentally with e.g. NMR spectroscopy, and then overlaps these structures in order to find common elements. For the second step it is common to model a structure as a tree, as introduced by Shapiro and Zhang, 1990 in their `RNAdistance` tool, and applied also in `RNAforester` [Höchsmann, 2005]. Other approaches use graphic vector representations [Liao and Wang, 2004] or the transformation of a possibly pseudoknotted structure into a linear sequence [Liu and Wang, 2006]. This category is explored to a much lesser extent [Pervouchine, 2018] and suffers from the problem that MFE structures often do not model the biologically relevant interactions.

Under a sequence-structure alignment we understand a structural information-guided alignment of RNA sequences [Tan et al., 2017]. The benefit is that both sequence and structure support the computation, as demonstrated in figure 2.3. However, considering structural information adds complexity to the problem of aligning sequences. The original algorithm for simultaneous alignment and folding by Sankoff, 1985 has the time complexity $\mathcal{O}(L^6)$ for the pairwise case with sequence length

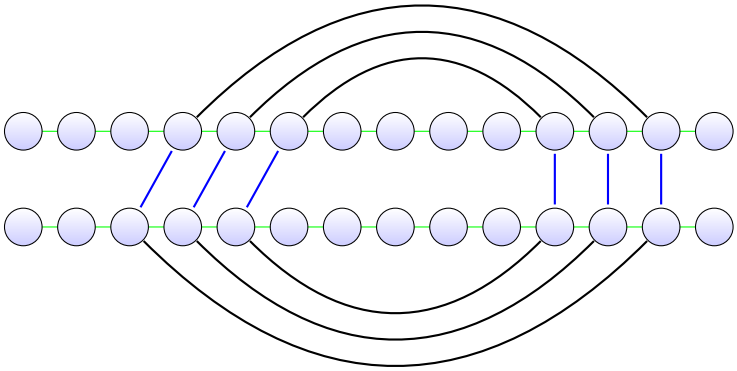


Figure 2.3: **A sequence-structure alignment.** The scoring function of this alignment considers equal nucleobases as well as matching secondary structure elements (blue lines).

L. The Sankoff algorithm applies dynamic programming and essentially integrates sequence alignment by Needleman and Wunsch, 1970 and structure alignment by Nussinov and Jacobson, 1980. The tool `LocARNA` [Will et al., 2012] reduces the time complexity to $\mathcal{O}(L^4)$ by using a progressive alignment scheme. A quadratic complexity is reached by the programs `SPARSE` (part of `LocARNA`) [Will et al., 2015] and `LaRA` [Bauer and Klau, 2005].

The most recent tool (as of May 2021) is `RNAmountAlign` [Bayegan and Clote, 2020], which uses mountain distance for pairwise structural pairwise alignments and runs in $\mathcal{O}(L^3)$ time for sequences of length L. Bayegan and Clote, 2020 demonstrate besides `RNAmountAlign` a good performance for `LocARNA` [Will et al., 2007] and `T-LaRA` [Bauer et al., 2007], which we investigate later on in this thesis. An overview of 71 tools or algorithms related to sequence-structure alignment is published by Lalwani et al., 2014.

2.2.1 Multiple alignment

In most cases, we need to align more than two sequences, and we denote such an alignment as *multiple sequence alignment* (MSA). In an optimal MSA the aligned nucleobases are maximally similar according to some specified criteria. Even when neglecting the structure, the computation of an accurate MSA is an NP-complete problem [Chatzou et al., 2016]. Adding the secondary structure conservation as an alignment criterion, the problem becomes even more difficult. All existing algorithms for multiple sequence-structure alignment (MSSA) use heuristics, and finding a good solution is still a challenging task [Chatzou et al., 2016, Warnow, 2021].

The Sankoff algorithm, which is the foundation of most of the algorithms described above, has a time complexity of $\mathcal{O}(L^{3n})$ and memory requirements of the order of $\mathcal{O}(L^{2n})$ for aligning n sequences of average length L [Sankoff, 1985], which

is prohibitive for most use cases. Therefore, different approximations have been implemented.

Some programs restrict the computation of the dynamic programming matrix to a band around the main diagonal. This limits the size and shape of the structure elements, but avoids computations, which are likely not in the optimal solution. However, this is problematic for the analysis of eukaryotic or virus RNA with long-range interactions, a field, which still lacks efficient computational methods [Pervouchine, 2018,Marz et al., 2014]. Tools with banded alignment include `Dynalign` [Mathews and Turner, 2002], `Foldalign` [Gorodkin et al., 1997], and `SCARNA` [Tabei et al., 2008].

The majority of tools, like `MARNA` [Siebert and Backofen, 2005], `PMcomp` [Hofacker et al., 2004], `LocARNA` [Will et al., 2007], `FoldAlignM` [Torarinsson et al., 2007], `T-LaRA` [Bauer et al., 2007], and `R-Coffee` [Wilm et al., 2008], use progressive multiple alignment to reduce the complexity of the MSSA problem. Progressive alignment is a heuristic developed originally for protein sequences by Feng and Doolittle, 1987 with the idea to trust the comparison of recently diverged sequences more than those that evolved in the distant past. Therefore, the approach uses the *once a gap — always a gap* policy, as changing gap choices later would increase the weight of more distantly related sequences. The algorithm computes in a first step $\frac{n(n-1)}{2}$ pairwise alignments from all combinational pairs of the given n sequences. With the help of a guide tree, which groups the most similar alignments together, the alignments are progressively combined to larger multiple alignments. The famous `ClustalW` tool [Thompson et al., 1994] implements this strategy for multiple sequence alignment.

The main problem of progressive alignment is that errors in the initial alignments are propagated to all subsequent steps. This can be overcome with consistency approaches, as the tools `T-Coffee` [Notredame et al., 2000] and `MAFFT` [Katoh et al., 2002] implement them. They are much more reliable tools for progressive multiple alignment, because they incorporate the information of all other sequences to the initial alignments, and therefore minimize the errors that are propagated to the next steps. The consistency paradigm can be applied to MSSA, e.g. the tool `T-LaRA` uses `LaRA` for pairwise sequence-structure alignment and combines these progressively with `T-Coffee`. The `X-INS-i` extension [Katoh and Toh, 2008] of `MAFFT` goes one step further and incorporates structural information in the progressive multiple alignment. Based on the structural pairwise alignments from `LaRA` or `SCARNA`, and the base pair probabilities from McCaskill's algorithm, it adds a so-called four-way-consistency score contribution to the progressive alignment, which favours base pair interactions of high probability in combination with a high pairwise similarity of the involved nucleotides.

2.2.2 The algorithm of LaRA

As described in the previous paragraph, `LaRA` computes a *pairwise* sequence-structure alignment. Here, we want to look deeper into the algorithm, as in part II of this

(a) An example input graph. (b) Interaction matches.

Figure 2.4: **LaRA's graph model for structural alignments.** The vertices are the nucleobases of the two sequences, the dashed lines are interaction edges and the solid lines represent the alignment edges. The figures are taken from Bauer and Klau, 2005.

thesis I present my re-implementation of the tool. Unless cited otherwise, the content of this subsection is based on Bauer and Klau, 2005.

The main idea of `LaRA` is to formulate an integer linear program (ILP) based on a graph representation of the sequence-structure alignment. With the ILP we aim to find the alignment, which maximizes the combined sequence-structure score.

Given two sequences x, y and their base pair probabilities $\mathcal{P}_x, \mathcal{P}_y$, we model a graph $G = (V, A \cup I)$. In the graph, the vertices V are the nucleobases of the two sequences x and y. The graph's vertices are connected by two distinct kinds of edges, as demonstrated in figure 2.4a:

1. A set A of *Alignment edges* (solid lines) that link vertices from sequence x with vertices from sequence y. An edge $(x_i, y_j) \in A$ models the alignment of x_i with y_j, i.e. the i-th character of the first sequence with the j-th character of the second sequence. Two alignment edges (x_a, y_c) and (x_b, y_d) are *in conflict*, if they touch or cross each other, i.e. $a < b \Rightarrow c < d$ or $b < a \Rightarrow d < c$ is violated. An alignment is a subset $\mathcal{A} \subseteq A$, where no alignment edges are in conflict.

2. A set I of *Interaction edges* (dashed lines) that represent intramolecular structural interactions, and therefore form connections among the vertices of a sequence. An edge $(x_i, x_j) \in I$ represents a base pair interaction of the i-th and j-th nucleobase in sequence x. An ordered pair of interaction edges $(i_x, i_y) \in I^2$ with $i_x = (x_a, x_b)$ and $i_y = (y_c, y_d)$ is called an *interaction match*, if and only if the incident alignment edges (x_a, y_c) and (x_b, y_d) are realized by the alignment $\mathcal{A}$. Figure 2.4b shows an example of two (pseudoknotted) interaction matches.

For incorporating a score, we annotate the edges with weights. The weight w_l of an alignment edge l is the score of aligning the two incident nucleobases, which given by the *Ribosum65* matrix [Klein and Eddy, 2003]. This 5×5 matrix contains the score for all combinations of the four RNA nucleobases and a wildcard symbol. The score is a positive number for equal nucleobases (i.e. AA, CC, GG, and UU), and a negative number otherwise (e.g. AC, NU, etc.).

The weight w_{lm} of an interaction match realized by two alignment edges $l = (x_a, y_c)$ and $m = (x_b, y_d)$ is derived from the respective base pair probabilities $\mathcal{P}_x(a, b)$ and $\mathcal{P}_y(c, d)$. In order create an additive scoring scheme from the probabilities, we transform them to the logarithmic space [Bauer et al., 2008]:

$$w_{lm} = \frac{1}{2} \ln \left(\frac{\mathcal{P}_x(a, b)}{p_{\min}} \right) + \frac{1}{2} \ln \left(\frac{\mathcal{P}_y(c, d)}{p_{\min}} \right) \tag{2.3}$$

The value $p_{\min} = 0.003$ is the minimum probability that is considered to be an interaction, i.e. for any entry $\mathcal{P}(i, j) < p_{\min}$ there is no corresponding interaction edge. Note that w_{lm} and w_{ml} are distinct variables (with the same value), as the pair (l, m) of an interaction match is ordered.

The graph is subject to some constraints in order to model a *valid* alignment: Every vertex is incident to at most one interaction edge (2.5). The end nodes of each interaction match must be part of the alignment $\mathcal{A}$ (2.5 and 2.7). No alignment edges are in conflict (2.6). Each two related interaction matches (where the pair is inverted) must be both either present or absent (2.7). These constraints can be modelled as an ILP, where the objective function aims to maximize the score achieved by the weights of the interaction and alignment edges, as equation (2.4) shows.

$$\max \sum_{m \in A} \sum_{l \in A} w_{lm} b_{lm} + \sum_{m \in A} w_m b_m \tag{2.4}$$

$$\text{s. t.} \sum_{m \in A} b_{lm} \leq b_l \qquad \forall l \in A \tag{2.5}$$

$$\sum_{l \in \mathcal{X}} b_l \leq 1 \qquad \forall \text{ sets of crossing alignment edges } \mathcal{X} \tag{2.6}$$

$$b_{lm} = b_{ml} \qquad \forall l, m \in A, l < m \tag{2.7}$$

$$b \in \{0, 1\} \tag{2.8}$$

The boolean variable b_l has value 1, if and only if $l \in \mathcal{A}$. Analogously, $b_{lm} = 1$ if and only if the alignment edges l and m realize an interaction match.

Because solving the ILP is an NP-hard problem, Bauer and Klau, 2005 use Lagrange Relaxation to simplify the *original problem*. Practically, this means dropping the constraint of equation (2.7) and penalizing its violation in the objective function:

$$\max \sum_{m \in A} \sum_{l \in A} w_{lm} b_{lm} + \sum_{m \in A} w_m b_m + \sum_{m \in A} \sum_{\substack{l \in A \\ l < m}} \lambda_{lm} (b_{lm} - b_{ml}) \tag{2.9}$$

The λ_{lm} variables in equation (2.9) are Lagrange multipliers, and the task is to find multipliers that provide the best upper bound to the original problem. With $\lambda_{ml} = -\lambda_{lm}$ for $l < m$ and $\lambda_{ll} = 0$ we can note the *relaxed problem* as ILP with constraints 2.5, 2.6, 2.8, and the following objective function.

$$\max \sum_{m \in A} \sum_{l \in A} (\lambda_{lm} + w_{lm}) b_{lm} + \sum_{m \in A} w_m b_m \tag{2.10}$$

The relaxed problem is a pairwise sequence alignment problem with a position-specific score. A formal proof can be found in Bauer et al., 2008. In short, for each alignment edge we can compute the maximum score that it can contribute to the alignment by choosing the outgoing edge with maximum weight w_{lm} that is not in conflict and add the edge's own weight w_l. The score of each alignment edge gives an entry in a position-specific score matrix (PSSM), which is then used by a global alignment algorithm, e.g. by Needleman and Wunsch, 1970.

We denote the resulting alignment the *relaxed* solution, because it may violate the constraint 2.7. Its score z_U is an upper bound for the optimal *valid* solution, because the computed alignment is optimal with respect to fewer constraints.

If for all pairs of alignment edges l and m the equation $b_{lm} = b_{ml}$ holds, then we have found the optimal valid solution to the original problem. Otherwise, some interaction edges contradict each other.

Given the fixed set $\mathcal{A}$ of active alignment edges, we have to find a subset of interaction edges such that each nucleotide is paired with at most one other nucleotide and the interactions have the maximum weight. This is a general maximum weighted matching problem [Bauer and Klau, 2005], which is discussed in section 4.4. The result is a *valid* structural alignment and its score z_L is a lower bound for the solution of the original problem.

Overall, `LaRA` iteratively solves the relaxed problem by employing iterative subgradient optimization. The Lagrange multipliers λ are incorporated into the scoring matrix. In each iteration, from the alignment a new lower bound is computed by finding the best structural interactions of this alignment and the Lagrange multipliers are updated accordingly. The solutions get increasingly better through the iterations and the bounds z_U and z_L provide a quality guarantee after any number of iterations. When the bounds coincide, the optimal solution has been found.

2.3 RNA homology search

As we have seen in chapter 1, one of the major challenges with lncRNA is linking sequence and function. Sequence alignment requires relatively long regions of high sequence conservation, which lncRNAs do not have. Thus, a better way is the detection of rather short conserved motifs within long and rapidly evolving transcripts. [Constanty and Shkumatava, 2021]

In the past, a genome-wide annotation of non-coding RNA was restricted to homologs of already identified RNA families [Griffiths-Jones, 2007]. There are specialized algorithms that spot a particular RNA family, like `tRNAscan-SE` [Chan and Lowe, 2019] and `ARAGORN` [Laslett and Canback, 2004] for tRNA, or `miRscan` [Lim et al., 2003] and `miRseeker` [Lai et al., 2003] for miRNA. Although they work really well and efficient within their scope, it is tedious to invent new algorithms for each RNA family, and it is impossible to discover new families with them.

Another class of algorithms is more general and can be applied to all families of RNA. These require known properties of the families, like patterns, motifs, or

profiles. For instance, `PatSearch` [Grillo et al., 2003] and `Structator` [Meyer et al., 2011] work with patterns, `RNAMotif` [Macke et al., 2001] takes structural motifs as input, and `Infernal` [Nawrocki and Eddy, 2013] as well as `ERPIN` [Lambert et al., 2004, Prince et al., 2022] work with profiles.

According to Macke et al., 2001, the benefits of structural motifs over pattern-based descriptors are the ability to incorporate context information. A motif allows for different variations in sequence or loop length with individually assigned scores.

For user-defined patterns and motifs, the specification of RNA structure and sequence constraints must be defined in one of several descriptor languages, which are often specific to a particular tool. Beyond their need for a-priori knowledge about the structure of the RNA family, a good understanding of the exact requirements is necessary, since too subtle or tight constraints can easily disturb the balance between specificity and sensitivity [Gautheret and Lambert, 2001].

The advantage of profile-based approaches is that they generate statistical profiles automatically from a multiple (structural) alignment and thus do not need the descriptor language. A profile is a statistical model that incorporates scores for sequence and secondary structure. A group of such tools is based on Stochastic Context Free Grammars (SCFG). They derive their statistical model as sets of production rules with associated probabilities [Gautheret and Lambert, 2001]. However, these tools cannot model pseudoknots and suffer from high computational demands.

One such SCFG-based tool is `Infernal`, which is very successful and probably one of the most frequently used tools in the field of RNA homology search. It computes consensus secondary structure profiles, so-called covariance models (CMs). These can be used for homology searches in a sequence database, or for computing novel alignments [Nawrocki et al., 2009]. Rfam, the most well known database of RNA families, is based on these covariance models, and it contains as well the genomic transcripts that have been found by applying `Infernal` on the CMs [Griffiths-Jones et al., 2003].

Since the CM-based method is very slow due to the expensive creation and calibration of the model, there have been efforts to increase the speed, e.g. by filtering the relevant sequences [Sun et al., 2012]. In part III of this thesis I present a novel program called `MaRs`, which is capable of searching an RNA family in a genomic database, and it is orders of magnitude faster than `Infernal` (see figure 12.6).

Instead of CMs, the sequence-structure information can also be represented as a collection of stem loops. A stem loop consists of a hairpin loop and an enclosed stem region that may contain interior or bulge loops (compare section 1.3 and figure 9.2). The advantage of the stem loop representation is that it can naturally deal with pseudoknots and that the overall complexity is reduced due to splitting the molecule into independent units of conserved structures [Sorescu et al., 2012].

For finding transcripts of an RNA family in a large genomic data set, the methods can be grouped into two categories: Tools of the first category scan the database, apply various filters and compute matches, e.g. `Infernal` [Nawrocki and Eddy, 2013], `LocARNAscan` [Will et al., 2013], or `tRNAscan-SE` [Chan and Lowe, 2019]. The second group of tools uses an index of the data set, like a suffix array in `RaligNAtor` [Meyer

et al., 2013], or affix trees for bi-directional search as in `Structator` [Meyer et al., 2011] or Schnattinger et al., 2012. Also `MaRs` is part of this second category, since it employs a bi-directional FM-index [Ferragina and Manzini, 2000, Lam et al., 2009], which has been implemented in `SeqAn` by Pockrandt, 2015 and was improved with so-called Enhanced Prefixsum Rank dictionaries [Pockrandt et al., 2017]. The advantage of this FM-index implementation is that it has a much lower memory consumption compared to the affix tree, and the results in section 12.1 show that the index creation and the search run faster by orders of magnitude.

Part II

LaRA 2: Sequence-structure alignment of RNA

3 Reading input and computing a structure annotation

LaRA 2 works on a set of at least two RNA sequences with *structure annotation*. An RNA sequence is a string of L characters over the RNA alphabet $\alpha = \{A, C, G, U, N\}$ where the characters represent the four nucleotides Adenine, Cytosine, Guanine, and Uracil, plus the wildcard for an unknown nucleobase, respectively. The structure annotation of a sequence x of length L is given as an $L \times L$ matrix $\mathcal{P}$, where the entry $\mathcal{P}(i, j)$ denotes the probability p that nucleotide x_i and nucleotide x_j form a base pair in the secondary structure of the RNA molecule.

For the sequence input, LaRA 2 accepts several formats from two categories: sequence files and secondary structure files. Sequence file formats are the ones that SeqAn 2 can parse, namely *Fasta*, *Fastq*, *Embl*, *Genbank*, *Raw*, *Sam*, and *Bam*. As file formats for secondary structure I have implemented *Connect*, *Stockholm*, *Dotbracket*, *Vienna*, *Ebpseq*, and *Bpseq*. The user specifies the parameter `-i` to pass a file in one of the mentioned formats to LaRA 2, which extracts the sequences and stores them as `Rna5` string, which is an efficient string implementation in SeqAn over α.

For the structure annotation input, LaRA 2 accepts three alternative ways, which are discussed in detail in the following sections.

1. The secondary structure format *Ebpseq* already contains the base pair probabilities for each sequence.

2. The user specifies the `-d` parameter to pass multiple dot plot files, from which the structure annotation and the sequence are extracted by LaRA 2. Dot plots can be retrieved as output from e.g. the `RNAfold` tool [Lorenz et al., 2011] with the partition function parameter (`-p`) enabled, which implements McCaskill's algorithm (see section 2.1.3 and McCaskill, 1990). As the dot plot contains the sequence already, an additional sequence input is not needed.

3. The structure input can be omitted and LaRA 2 computes the structure annotation internally, using the API of `RNAfold`.

3.1 The Ebpseq file format

With the purpose of integrating all kinds of different input data for RNA analysis into a single file, Gianvito Urgese and I have designed the *Ebpseq* file format. An

```
## G: General field with arbitrary information or comments for the file.
## S1: Name of the first sequence
## S2: Name of the second sequence
## S3: Name of the third sequence
## F1: Secondary structure, method 1
## F2: Secondary structure, method 2
## M1: Base pair probability matrix, method 1
## T1: Type of biologically validated data (e.g. SHAPE, DMS, CMCT).
## T2: Another type of biologically validated data (e.g. SHAPE, DMS, CMCT).
# S1 T2
# I     NT      QU      R2      RE2     F1      F2      M1
1       A       H       3.4552  1.1760  8       5       <5/0.003 | 8/0.87>
2       G       {       46.9128 13.8533 7       4       <5/0.03 | 7/0.4>
3       U       #       0.1740  0.2738  6       6       <6/0.5>
4       C       !       0.0000  0.0000  0       2       <>
5       C       7       0.0000  0.0000  0       1       <1/0.003 | 1/0.03>
6       G       T       0.0279  0.0161  3       3       <3/0.5>
7       U       @       0.0997  0.0575  2       0       <2/0.4>
8       C       g       0.0000  0.0000  1       0       <1/0.87>
# S2 T2
# I     NT      QU      R2      RE2     F2      M1
1       A       @       3.4552  1.1760  8       <5/0.003 | 8/0.87>
2       G       {       16.9128 1.8533  7       <5/0.03 | 7/0.4>
3       C       #       0.1740  0.2738  6       <6/0.6>
4       C       !       0.0000  2.0220  0       <>
5       C       &       0.4300  0.0110  0       <1/0.003 | 1/0.03>
6       A       T       0.0279  0.0161  3       <3/0.6>
7       U       @       0.0997  0.0575  2       <2/0.4>
8       C       1       0.0000  0.0000  1       <1/0.87>
# S3 T1 T2
# I     NT      QU      R1      RE1     R2      RE2     F1      M1
1       A       ?       3.4552  1.1760  0.7127  0.0863  8       <4/0.002 | 5/0.3>
2       G       '       46.9128 13.8533 0.3916  0.1568  4       <4/0.6 | 7/0.3>
3       U       5       0.1740  0.2738  1.5855  0.2431  6       <6/0.8>
4       C       !       0.7700  2.4500  0.3236  0.6132  2       <1/0.002 | 2/0.6>
5       C       @       0.0000  0.0000  1.6117  0.0000  1       <1/0.3>
6       G       \       0.0279  0.0161  0.4674  0.0000  3       <3/0.8>
7       U       @       0.0997  0.0575  0.3222  3.9352  0       <2/0.3>
```

Listing 3.1: **The versatile Ebpseq file format.** It consists of a header for textual descriptions of the sequences, structures and methods to obtain the data, and one or multiple records that contain values in tabular form, including e.g. the RNA nucleobase, a quality code, reactivity data, fixed structures, and base pair probabilities for each sequence position.

Ebpseq file consists of a header, followed by one or multiple data records, as shown in the example file in listing 3.1.

In the header various data sources are defined and linked to their identifiers, which in turn can be referenced from the individual records. For instance, `S1` is the identifier of the first sequence, which has a given name (e.g. extracted from a *Fasta* header) and is referenced in the first record. In the same way it is possible to specify with which method a fixed secondary structure (`F`) or base pair probability matrix (`M`) has been retrieved, and which type of experimental validation (`T`) has been used. Each identifier must be unique, and each group of identifiers can have arbitrary many elements or be omitted.

The *Ebpseq* records have a tabular structure with one line per nucleobase, due to their design as an extension of the *Bpseq* format. The leftmost three columns hold therefore serial numbers as index, the nucleobases, and (optional) sequencing qualities, respectively. Further columns can be present according to the availability of data, and they can vary among different records.

An `F` column contains a secondary structure and holds the respective index of the paired base. In case a base is unpaired, the value is 0 (which does not cause conflicts, as the indices in the first column start from 1). The base pair probabilities in an `M` column are represented as list enclosed in angle brackets and divided with a vertical bar (`|`). Each list element is a pair consisting of the partner index and an assigned probability. An empty list represents a site without interaction.

The `R` and `RE` columns are linked to the `T` identifier. They are meant to represent the contents of RDAT files, i.e. the reactivity values (`R`) and reactivity error (`RE`) of RNA structure mapping experiments, like SHAPE. The reactivity is a likelihood estimate whether the respective site is paired or not, accompanied by the standard error between the samples used. Each record can be tagged with zero, one or more `T` identifiers (see the example file) and their indices must be equal to the used `R` and `RE` indices. These data can be used to add constraints or weights to the structural interactions.

The *Ebpseq* file format can be used as a database to collect all types of structural information of related sequences. It can be used as single-file input for `LaRA 2`. I implemented the format for reading and writing in SeqAn, where it is publicly available since version 2.3.0.

3.2 Parsing dot plot files

A dot plot file is the result of computing the partition function of a sequence with the `RNAfold` program [Lorenz et al., 2011] (`-p` flag enabled). The program generates one file per sequence and the filename consists of the sequence name and the suffix `_dp.ps`. Besides the sequence, it contains the base pair probabilities of possible interactions.

Figure 2.2 on page 15 shows a dot plot file that is opened with a Postscript viewer. It shows a matrix which is labelled in both directions with the same sequence. The

```
/sequence { (\
UAGGACACUCUCUAUCAGAAUGGAUGUCUUGCUGUCAUAACAGAUAGAGAAGGUUGUGGCAGACC\
) } def
```

```
%start of base pair probability data
1 14 0.003357526 ubox
1 31 0.804069207 ubox
1 58 0.059768145 ubox
1 63 0.004190175 ubox
2 9 0.004788885 ubox
2 13 0.003736190 ubox
2 29 0.017668299 ubox
2 30 0.948162637 ubox
2 57 0.066164678 ubox
3 8 0.005870257 ubox
```

Listing 3.2: **Excerpt of a dot plot file when opened in a text editor.** From the Postscript code it is possible to extract the sequence and the base pair probabilities.

probability of an intramolecular interaction of a pair of nucleobases is shown in the respective cell with a dot, where in the upper triangular matrix the size of the dot corresponds to the base pair probability. The lower triangular matrix shows the optimal structure, as computed with the McCaskill algorithm (see section 2.1).

When the same dot plot file is opened in a text editor, the underlying Postscript data can be accessed, as demonstrated in listing 3.2, which is an excerpt of the file that contains figure 2.2.

The sequence can be located in the file by searching for the `/sequence` variable. It may contain a newline character preceded by a backslash at multiple positions, which are skipped in the parsing process. The characters are interpreted as RNA nucleobases and stored efficiently as `Rna5` string. Characters that are not in α are rejected.

The base pair probabilities are located in the lines that end with `ubox`, which encode the upper triangular matrix of the dot plot. After removing the `ubox` suffix, each of these lines can be interpreted as a triple of two integral numbers and one floating point number. The integral numbers represent the indices of the nucleobases within the sequence that form a pair with the probability given by the floating point number. For instance, in the example file, base number 2 forms a pair with base number 30 with a probability of 94.8%.

I implemented the file format in `LaRA 2` for reading. If a user provides dot plot files, `LaRA 2` extracts the sequences with the respective base pair probabilities as input.

3.3 Computing a structure annotation

If for any input sequence there are no base pair probabilities given, `LaRA 2` computes them with the `RNAlib` library of the `ViennaRNA` package [Lorenz et al., 2011], which is an implementation of the algorithm by McCaskill, 1990. It produces the same

```
1 find_library (VIENNA_RNA_LIB libRNA.a)
2 find_path (VIENNA_RNA_PATH NAMES ViennaRNA/part_func.h)
3 if (VIENNA_RNA_LIB AND VIENNA_RNA_PATH)
4     add_definitions (-DVIENNA_RNA_FOUND)
5     target_link_libraries (lara PUBLIC ${VIENNA_RNA_LIB})
6     target_include_directories (lara SYSTEM PUBLIC ${VIENNA_RNA_PATH})
7 endif ()
```

Listing 3.3: **Linkage of the RNAlib as an optional dependency.** If it is available, the macro `VIENNA_RNA_FOUND` is defined (line 4), which can be queried in the program code.

```
1 int length = seqan::length(sequence);
2 init_pf_fold(length);
3 float energy = pf_fold(seqan::toCString(sequence), nullptr); // fills the arrays pr and iindx
4
5 for (int i = 1; i <= length; ++i)
6     for (int j = i + 1; j <= length; ++j)
7         double prob = pr[iindx[i] - j];
```

Listing 3.4: **Computation of the base pair probabilities in LaRA 2.** The main work is done by the `pf_fold` function of the `RNAlib`, which computes the partition function and writes the base pair probabilities to the `pr` array, which is publicly defined in `RNAlib`. An additional index array `iindx` helps to interpret the `pr` data as shown in line 7.

results as if the user produces dot plot files with `RNAfold` and passes them to `LaRA 2`. The library has a `C` interface (which is accessible from C++) and the current version of April 2021 is 2.4.18. The use of the `RNAlib` API has two advantages: Firstly, the production of an output file per sequence, their storage on a hard drive, and the subsequent extraction of the relevant data in `LaRA 2` requires time and disk space that can be saved. Secondly, passing a single sequence file to `LaRA 2` with possibly hundreds of sequences is much more handy than specifying the filenames of possibly hundreds of dot plot files.

As shown in listing 3.3, I linked the `RNAlib` as an optional dependency into `LaRA 2`. The advantage is that `LaRA 2` can be used even if the library is not available, however the computation of a structure annotation is then not possible. If in this case the user passes only sequence files, an error message is shown, which clarifies that the library is missing.

Two functions of the `RNAlib` are being used in the program code of `LaRA 2`, namely `init_pf_fold` and `pf_fold`. The first one allocates space dependent on the length of the given sequence x. The latter performs the actual computation of the partition function and returns the minimum free energy of the optimal structure, which could be queried with the second parameter of `pf_fold`, but is not needed here. The base pair probabilities are contained in a public variable `pr`. The probability $\mathcal{P}(i, j)$ that x_i and x_j form a pair can be accessed from `pr` with help of the index array `iindx`, as shown in line 7 of listing 3.4.

The full API of the `RNAlib` can be accessed on https://www.tbi.univie.ac.at/RNA/ViennaRNA/doc/html/index.html (01.06.2021). Since I have implemented `LaRA 2`, the interface of the `RNAlib` has advanced, and the described `pr` and `iindx` arrays are marked deprecated. Therefore, I suggest updating `LaRA 2` to the newer API in future developments, namely using the function `vrna_pf_fold`, which has the same interface as `pf_fold`, except that it takes as third argument a pointer to a list, where the base pair probabilities are stored. The new implementation is thread-safe and the computation of structure annotations in `LaRA 2` can thus be parallelized over the number of sequences.

4 Solving the structural alignment

After parsing the input files as explained in the previous chapter, LaRA 2 compiles a set of all pairwise sequence indices. All these sequence pairs are independently treated by LaRA 2 in order to compute structural alignments, and afterwards they are passed to a progressive alignment method for combining them to a multiple alignment. Therefore, in the following we are looking at computing a single, individual pairwise structural alignment, and for their parallel computation and multiple alignment please read chapter 5 and chapter 6, respectively.

4.1 Filtering relevant interactions

Given two input sequences x and y of length L and M, there are $L \cdot M$ possible alignment edges that connect the characters of sequence x with the characters of sequence y. In order to reduce the memory footprint of the program, we want to filter these edges to the most relevant ones.

A global sequence alignment with affine gap costs can be computed with the algorithm by Gotoh, 1990. Affine gap costs take into account that a gap has an initialization cost as well as an extension cost, such that a single long gap is preferred over multiple short gaps. As scores LaRA 2 uses by default the gap open score -6, gap extension score -2, and the *Ribosum65* scoring matrix (see page 19). The Gotoh algorithm is a dynamic programming algorithm (as introduced with the Nussinov algorithm in section 2.1), which computes matrices of size $(L+1) \times (M+1)$ that allow us to retrieve the score of the optimal alignment of all sequence prefixes $(x_1, x_2, \ldots, x_i)$ and $(y_1, y_2, \ldots, y_j)$ with $1 \leq i \leq L$ and $1 \leq j \leq M$.

Applying the Gotoh algorithm on the reversed sequences $(x_L, x_{L-1}, \ldots, x_1)$ and $(y_M, y_{M-1}, \ldots, y_1)$ computes the matrices that allow us to retrieve the score of the optimal alignment of all sequence suffixes $(x_i, x_{i+1}, \ldots, x_L)$ and $(y_j, y_{j+1}, \ldots, y_M)$ with $1 \leq i \leq L$ and $1 \leq j \leq M$.

Summing up the prefix score, suffix score and the *Ribosum65* score of a cell (i, j), we retrieve the maximum possible score s_{ij} that an alignment through (i, j) can achieve. An example table of these maximum scores for each cell is shown in table 4.1. We observe that an optimal alignment is represented in the table as a path of optimal scores with $s_{ij} = s_{\text{opt}}$ from the top left corner to the bottom right. A second observation is that the more a cell deviates from the optimal alignment path, the smaller is its score. LaRA 2 has a suboptimality parameter u with default value $u = 40$, which influences how strict the edge filter is. The implemented rule allows for each (i, j) pair the creation of an alignment edge $l = (x_i, y_j)$ only if $s_{ij} \geq s_{\text{opt}} - u$.

	C	A	G	A	A	A	C	C	U	G
C	4.5	-7.8	-12.0	-16.7	-22.1	-27.5	-30.7	-35.5	-41.5	-47.5
A	-0.3	4.5	-6.2	-9.7	-15.1	-20.4	-27.5	-32.3	-36.9	-41.7
G	-6.6	1.3	4.5	-6.2	-11.6	-17.0	-22.8	-27.6	-31.7	-35.0
A	-11.3	-4.3	3.2	4.5	-4.9	-10.3	-17.3	-22.1	-26.7	-31.5
A	-16.7	-9.7	-6.2	4.5	4.5	-4.9	-11.9	-16.8	-21.3	-26.1
A	-22.1	-15.1	-11.6	-4.9	4.5	4.5	-6.5	-11.4	-15.9	-20.7
A	-27.5	-20.4	-17.0	-10.3	-4.9	4.5	2.9	-6.0	-10.5	-15.3
C	-30.7	-27.5	-22.8	-17.3	-11.9	-6.5	4.5	2.9	-7.1	-12.8
C	-35.5	-32.3	-27.6	-22.1	-16.8	-11.4	-4.3	4.5	1.7	-8.3
U	-41.5	-36.9	-31.7	-26.7	-21.3	-15.9	-10.4	-5.5	4.5	0.2
G	-47.5	-41.7	-35.0	-31.5	-26.1	-20.7	-16.3	-12.6	-7.0	4.5

Table 4.1: **Maximum possible score of each edge.** This table is used by the edge filter to select the best alignment edges for the structural alignment. The optimal sequence alignment with $s_{opt} = 4.5$ is marked in green. The filter removes all edges, whose score differs at more than u from s_{opt} (shown in red for $u = 40$).

Since the `SeqAn` implementation of the Gotoh algorithm does not permit access to the underlying matrices and the algorithm is simple enough, I implemented it for `LaRA 2`. The file `edge_filter.hpp` contains the code that is needed for deciding which alignment edges are created and which are omitted.

Therefore, I implemented a class `PairwiseGotoh`, which on creation computes the necessary tables and provides functions to query the prefix score for a position (i, j) and the overall optimal score s_{opt}.

The public function `generateEdges`, which is shown in listing 4.1, uses two instances of the `PairwiseGotoh` class: one for the forward and one for the reversed sequences. It iterates once through the matrix of possible edges and determines, whether s_{ij} is equal to or larger than the threshold $s_{opt} - u$. The run time and memory of the edge filter have the complexity $\mathcal{O}(L \cdot M)$.

The computation of two alignments prior to the actual structural alignment may raise the question, whether the effort pays off. In fact, sequence alignment is a much easier problem than structural alignment and the runtime of the edge filter is usually less than 1% of `LaRA 2`'s overall runtime. Unless the given sequences have no structure at all, the time spent with the edge filter already amortizes in the first two iterations of structural alignment. Therefore, the efficiency of the filter is clearly dependent on the number of iterations, which is unknown a-priori. Furthermore, the edge filter is generally more efficient with longer alignments, since it is able to prune more edges.

In comparison to a banded alignment, where we limit the alignment to cells with a fixed maximal distance from the main diagonal, this edge filter has two advantages: it can react much better to different regions of similarity, and it would be quite

```
std::vector<bool> generateEdges(seqA, seqB, scoreMatrix, u)
{
    // generate reverse sequences
    seqan::ModifiedString<seqan::Rna5String const, seqan::ModReverse> reverseA(seqA);
    seqan::ModifiedString<seqan::Rna5String const, seqan::ModReverse> reverseB(seqB);

    // two instances of Gotoh algorithm
    PairwiseGotoh forward(seqA, seqB, scoreMatrix);
    PairwiseGotoh backward(reverseA, reverseB, scoreMatrix);

    // length of the sequences
    size_t const lenA = seqan::length(seqA);
    size_t const lenB = seqan::length(seqB);

    std::vector<bool> edges{lenA * lenB, false};
    int const threshold = forward.getPrefixScore(lenA, lenB) - u;

    for (size_t a = 0; a < lenA; ++a)
    {
        for (size_t b = 0; b < lenB; ++b)
        {
            if (forward.getPrefixScore(a, b)
                + seqan::score(scoreMatrix, seqA[a], seqB[b])
                + backward.getPrefixScore(lenA - a - 1, lenB - b - 1) >= threshold)
            {
                edges[a * lenB + b] = true;
            }
        }
    }
    return edges;
}
```

Listing 4.1: **Implementation of the edge filter.** The `PairwiseGotoh` class has a function `getPrefixScore`, which accesses the internal tables of the algorithm and returns the alignment score until the given position in constant time.

hard to set a good band distance a priori. The threshold value for the edge filter is dependent on s_{opt}, the optimal alignment score. This makes the filter very flexible for alignments of various length or sequence identity. The default value of $u = 40$ has been chosen as the smallest value such that it does not negatively influence the alignments in the Rfam database [Kalvari et al., 2018]. Since this is a diverse dataset of many RNA families, I assume that this value is suitable for general use.

4.2 Edge management

This is a short section about how edges are stored and referenced in `LaRA 2`, since they are a central part throughout the whole program.

For alignment edges, in `LaRA 2` so-called *lines*, there is a class `EdgeManager` that maintains a mapping between the position pair within the original sequences and the internal `edgeId`, which is based on the reduced number of edges from the edge filter (see previous section). The following operations are performed on average in constant time:

- `edgeId(source, target)`: retrieve the edge identifier from sequence positions
- `source(edgeId)`: retrieve the position within the first sequence
- `target(edgeId)`: retrieve the position within the second sequence
- `nonCrossing(edgeId1, edgeId2)`: return true if the two edges do not cross each other, false otherwise

Using an integral number to identify an edge has several advantages, e.g. a structural alignment is stored efficiently as a vector of `edgeId`, or a boolean vector can be used as a constant-time lookup table for whether an edge is part of an alignment. Most importantly, it simplifies the definition of interaction edges.

Structural interaction edges are stored as a vector of hash maps. Each `edgeId` is assigned a hash map containing potential partner edges (their identifiers are the hash keys) with additional information. This information includes the structural score of the pair derived from the base pair probability matrix, a unique identifier number of the pair (used after the matching step), and a pointer into the priority queue to allow constant-time updates of the queue.

A priority queue for a given line is a set of interaction partners, which consists of the profit score of realizing the interaction, and the `edgeId` of the partner line. Since the queue is sorted by score, we can access in constant time the interaction partner of a line with maximal profit and iterate through the set in descending order. Note that the profit score is initialized with the sum of structure score and sequence score, but opposed to those it will change in each iteration according to the Lagrange adjustments after the matching step. These changes may of course alter the order of the queue elements.

The score matrix for the structural alignment contains a value for each possible line. These values are set to the maximal profit scores given by the leading elements of the priority queues. After each iteration the values are updated accordingly and this is the key mechanism how `LaRA 2` incorporates the structure information and shifts the alignment towards a valid matching of interaction edges.

4.3 Alignment with position-specific score

We have seen in section 2.2.2 that the relaxed structural alignment problem is a pairwise sequence alignment with position-specific score. Fortunately, `SeqAn` already implements a very fast pairwise sequence alignment routine by Rahn et al., 2018, which is capable of SIMD vectorization, multi-threading, and affine gap scores. Of course, I wanted to use this implementation for `LaRA 2`. However, the `SeqAn` implementation lacked the ability of using a position-specific score matrix (PSSM), which is necessary for including the structural information.

The goal was therefore to implement an extension for this alignment algorithm, namely a new template instance for `seqan::Score`, which can answer the score

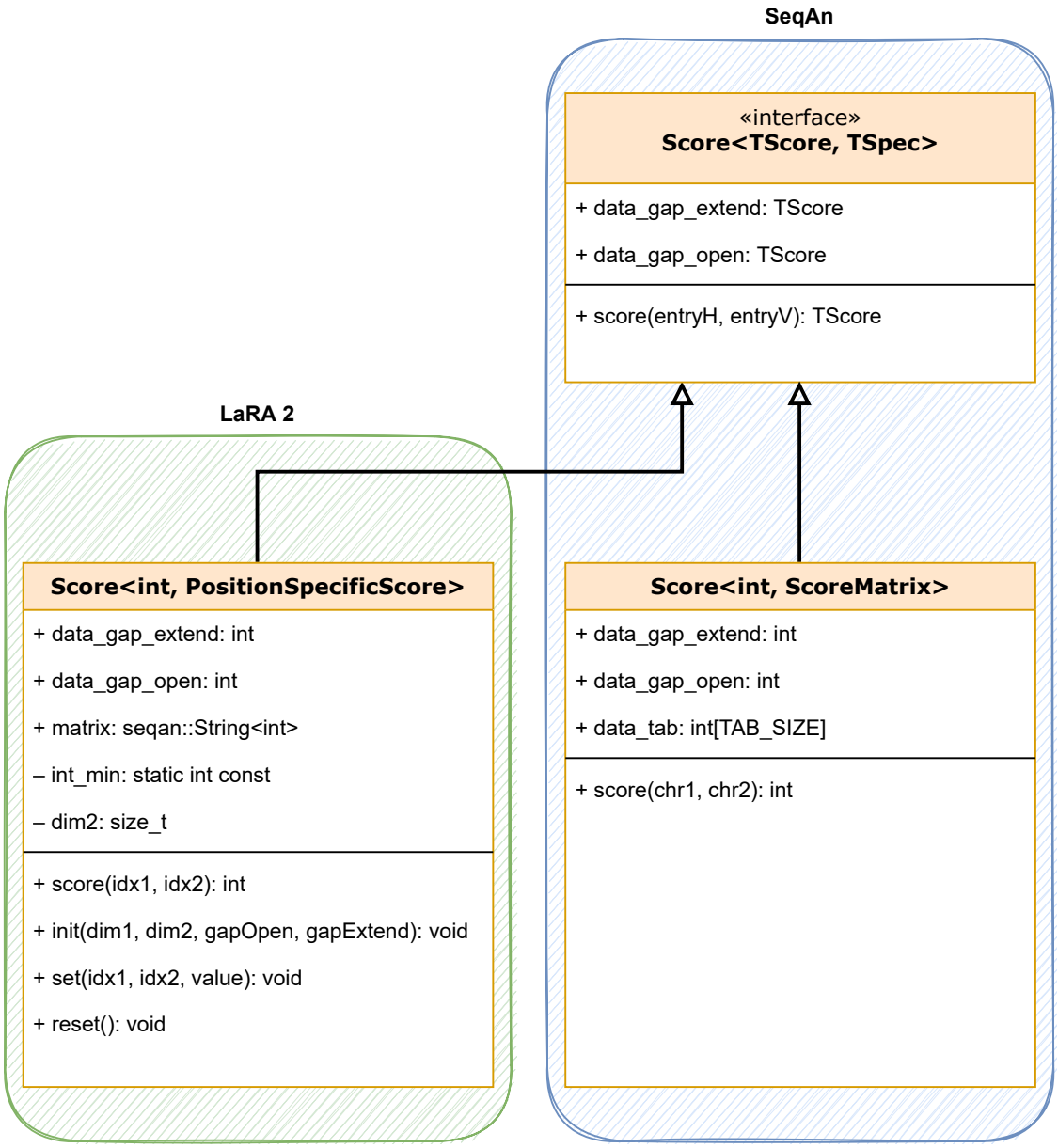


Figure 4.1: **Comparison of the score class interfaces.** The score class in `LaRA 2` implements the SeqAn class interface and extends it for position specific score with functions for initializing and resetting the whole matrix or for updating single values. For comparison, on the right there is the class definition for the score matrix implementation, which uses as matrix a fixed array of size `TAB_SIZE` $= |\alpha|^2$, where $|\alpha|$ is the size of the sequence alphabet.

queries of the algorithm. Unfortunately, the score queries of the template class are based on sequence characters, e.g. `score('C', 'C')`, which does not allow the determination of their positions (a.k.a. index) inside the sequences.

Since the sequence characters themselves are not relevant for the algorithm, but rather the indexing inside a sequence in order to determine the score, LaRA 2 creates two index sequences $(0, 1, 2, \ldots, L-1)$ and $(0, 1, 2, \ldots, M-1)$ and passes them to the alignment algorithm instead of the real sequences of length L and M. This was possible through a relaxation of the SeqAn interface such that it accepts sequences of integral numbers. With this trick, the score queries are based on the positional index, e.g. `score(23, 24)`, and it becomes possible to implement a new class template instance `score<int, PositionSpecificScore>` that returns the correct score.

The new score class contains an $L \times M$ matrix (linearized as `seqan::String`) for the position-dependent score, as well as two values for the gap open and gap extension scores. In LaRA 2, each score value is stored as 32 bit signed integer. Initial floating point numbers, e.g. from the *Ribosum65* score matrix or user parameters, are multiplied with the constant factor 8192 and converted to integer. The factor is chosen as a power of two, such that it ensures three digits of precision and provides still enough range $[-262\,144 \ldots 262\,144)$ for the original (untransformed) score.

The initialization value of each entry in the PSSM should be $-\infty$, which can be approximated with int_{min} = `std::numeric_limits<int32_t>::lowest()`, the minimum supported score value. However, with this value the alignment algorithm turned out to be not deterministic for some cases: Note that with the edge filter described above, some cells of the PSSM may remain $-\infty$ forever. I found out that during the evaluation of a neighbouring cell the algorithm occasionally generates an underflow, since a possibly negative score from the PSSM is added to the int_{min} value. In order to avoid any value underflows, internally $\frac{2}{3} \cdot \text{int}_{\text{min}}$ is used as value for $-\infty$.

Figure 4.1 visualizes how the score implementation fits into the SeqAn environment. Essentially, it is required to implement a function to retrieve the score and two members for the gap scores. The remaining functions and members are for maintaining and updating the matrix, since it has to be altered in each iteration. The private `int_min` member is the initialization value int_{min} as discussed above and used by the `init` and `reset` functions. The `dim2` member permanently stores the second sequence length in order to calculate the correct matrix index from a given position pair ($\text{index} = \text{dim2} \cdot \text{idx1} + \text{idx2}$) and is used by the `score` and `set` functions.

Now that we have seen how the position-specific score works and is implemented, let us zoom out a bit and focus on how it is used and what are the results of the alignment procedure.

Although LaRA 2 computes many alignments in which the PSSM has always a different content or size, at any time there exists only one such matrix (per thread). Since the matrix has always full $L \times M$ size, it would be expensive to delete and recreate it for each alignment (or even iteration). Instead, a function `updateScores` changes specifically the modified values between two iterations, which are on average 5% of the matrix values. When all alignments of a sequence pair are finished and the

next pair is loaded, the whole matrix is reset to $-\infty$. In order to guarantee that the allocated matrix is always large enough for the next alignment, the input sequences have been sorted by length and pairs are created with the longer sequence first, such that L is the length of the longest sequence and M the length of the second longest sequence. In subsequent alignments the matrix still has its original (maximal) size, but only a subset of cells may be accessed.

Likewise, the order of input sequence pairs is helpful for the creation of the index sequences. Instead of allocating and filling separate integer vectors for each sequence, `LaRA 2` stores only a single index sequence of length L, the length of the longest sequence. For any sequence x of length $L_x \leq L$, the prefix of length L_x of that pysical index sequence represents the corresponding index sequence of x. The prefix is computed with the function `seqan::Prefix`, which does not perform a copy and thus efficiently provides a (virtual) subsequence. A pair of such index sequence prefixes is used to invoke the alignment algorithm.

The returned result from the alignment algorithm are a pair of gapped integer sequences and the alignment score, of which the latter is an upper bound to the optimal solution's score. The gapped sequences must once be iterated linearly for two reasons: (1) The linked characters in the alignment represent edges that must be marked active (i.e. $b_l = 1$, see section 2.2.2) for consideration in the matching step. (2) At the same time we sum up the amount of (negative) gap score, since we need it for computing the lower bound solution.

4.4 Maximum weighted matching

A valid solution (the Lagrangian primal) of the original alignment problem is computed by applying a maximum weighted matching (MWM) algorithm on the interaction graph that is depicted in figure 4.2. The nodes of the interaction graph are the active lines, i.e. the edges that result from the alignment step. An undirected edge (shown as solid arc) is present between these nodes, where there exists an interaction match, i.e. corresponding structural interactions (shown as dashed arcs) in both sequences. See figure 2.4b on page 19 for a visualization of interaction matches.

A matching is an independent edge set, i.e. it does not have any common vertices. The MWM algorithm ensures that each nucleotide is incident to at most one structural interaction, while it chooses the interactions such that the sum of the edge weights is maximal. The weight of an edge between nodes l and m is the sum of the score of the two involved interaction matches $w_{lm} + w_{ml}$, which are derived from the base pair probabilities as previously shown in equation (2.3).

I have tested two different heuristics for MWM in general graphs: An extension of the Blossom algorithm by Edmonds, 1965a that is available in the Lemon Graph Library [Dezső et al., 2011], and a greedy approach with look-ahead strategy, which I have implemented in `LaRA 2`.

The method of the Blossom algorithm is the detection of cycles in the graph that consist of an odd number of edges, so-called blossoms. In a blossom with $2m+1$ edges

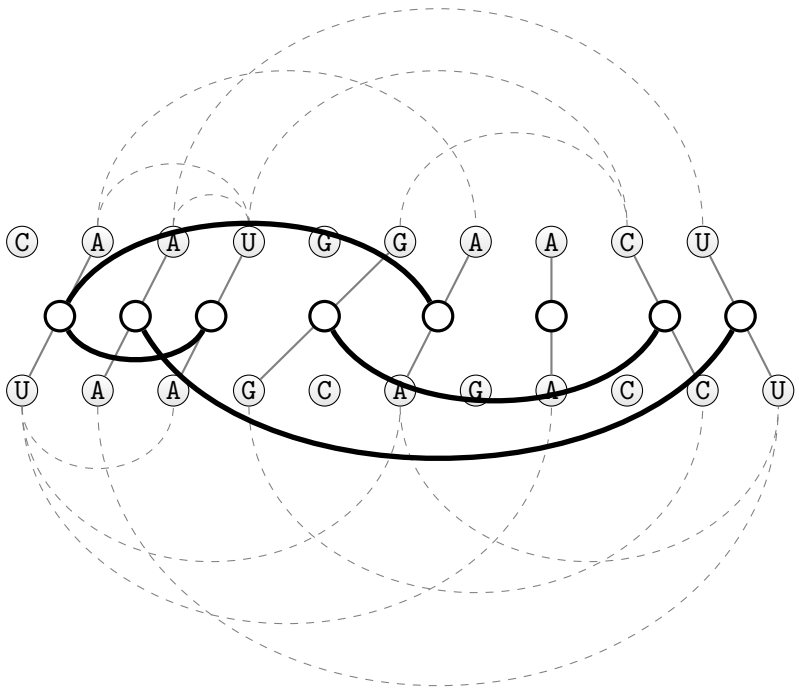


Figure 4.2: **Maximum weighted matching.** The algorithm selects the best valid interactions in order to compute a lower boundary for the optimal score. The matching property in the displayed interaction graph is not yet satisfied, because the leftmost node is incident to two interactions.

there are exactly m edges that are part of the matching, forming an alternating path. Thus, each blossom can be contracted into a single node and the search continues in the reduced graph. The running time is in $\mathcal{O}(EV^2)$ with E the number of edges and V the number of nodes [Edmonds, 1965b, Gerards, 1995].

For the greedy approach I generate a list of all edges sorted by their weight in descending order. Then I consider the maximum weighting k edges from the beginning of this list and perform an exhaustive search to find the maximum weighting set. The selected edges become part of the matching and all incident edges are excluded from the list. This process is repeated with the following k edges in the list, until the end of the list is reached.

An exhaustive search of k edges is performed by checking for each of the $\frac{1}{2}k(k-1)$ pairwise combinations whether they share a node with each other. If yes, then they are collected in a set of conflicting pairs, otherwise they immediately become part of the matching. The set of conflicting pairs is passed to the recursive `solveConflicts` routine. This routine chooses one edge and removes each entry from the list that involves this edge and calls `solveConflicts` on the reduced list. The algorithm keeps track of the edge scores, so it can compare at each stage the maximum score that results from including or excluding an edge.

The `solveConflicts` routine checks in the worst case 2^k alternative combinations of edges. It is called $\frac{E}{k}$ times, with E the number of edges in the graph. For each set

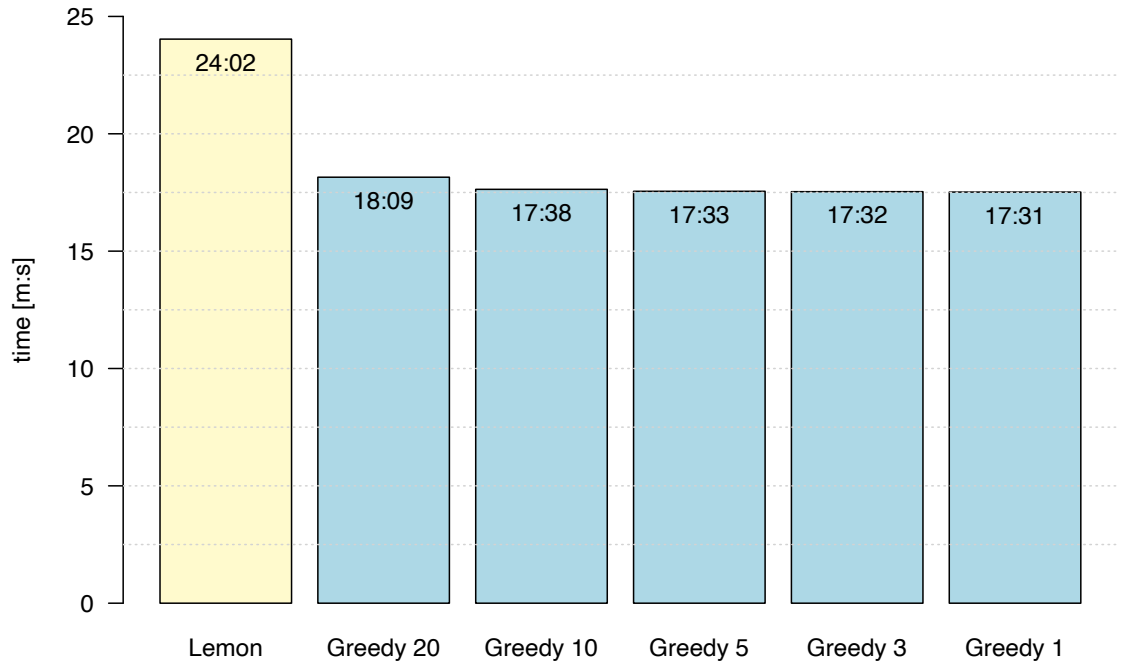


Figure 4.3: **Comparison of run time for maximum weighted matching.** The diagram shows the total run time of LaRA 2 in minutes for computing the 388 reference alignments of the BRAliBase 2.1 data set [Gardner et al., 2005]. The included methods are the blossom algorithm and the greedy approach with different look-ahead parameter $k \in \{20, 10, 5, 3, 1\}$.

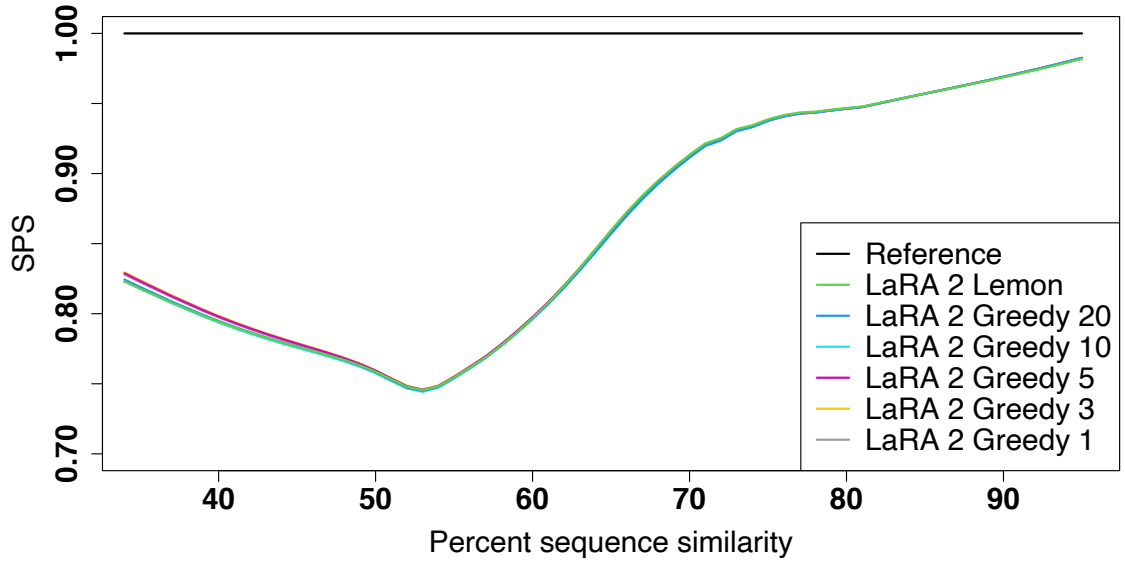


Figure 4.4: **Quality comparison for maximum weighted matching.** The diagram shows the sum of pairs score (SPS) of LaRA 2 for the BRAliBase 2.1 data set. Benchmark details are given in section 7.1.

of k edges, all incident edges are removed from the remaining list, which adds on average $\frac{E}{2}$ incidence checks of $4k$ comparisons at most. Thus, the whole algorithm executes at most $\frac{E}{k}(2^k + 2Ek)$ operations and with fixed parameter k the run time is in $\mathcal{O}(E^2)$.

A comparison of the total run time for LaRA 2 as shown in figure 4.3 reveals that the greedy approach results in a lower total run time compared to the blossom algorithm. Although the heuristics may produce suboptimal matchings, the resulting alignments do not loose quality (see figure 4.4), because LaRA 2 compensates the outcome with a few more alignment and matching iterations. For the default method in LaRA 2 I choose the greedy method with $k = 5$.

The score for the lower bound of the current LaRA 2 iteration is the sum of the weights of the edges that are part of the computed matching, plus the sequence alignment score. The highest score over all iterations together with the corresponding alignment is reported as the valid solution of the pairwise sequence-structure alignment problem.

4.5 Updating the Lagrange multipliers

The final topic we should discuss in the light of solving the structural alignment is the update of the Lagrange multipliers λ_{lm} such that the alignment converges towards the optimal solution. In LaRA 2 this is achieved with iterative subgradient optimization — the same method that is also employed in LaRA 1. Thus, I do not give many mathematical details on this method here and refer to Bauer and Klau, 2005 and section 2.2.2. I rather want to connect the implementations that were described in the previous sections, and discuss in detail the steps of an iteration. As an overview, each iteration in LaRA 2 performs the following sequence of steps:

1. compute global alignment with PSSM
2. determine a valid solution through maximum weighted matching
3. update the best upper and the best lower bound
4. terminate if the bounds are close enough or no iterations remain
5. halve the step size if there was no progress in 50 iterations
6. compute the next Lagrange multipliers
7. update the PSSM

The purpose of the iterations in LaRA 2 is to decrease the gap between the dual and primal solution. The dual solution is the result of the relaxed problem in equation (2.10), which is obtained in the alignment step as the sum of the scores for active alignment lines, incident structural interactions, and gaps. Since it results from the relaxed problem and is possibly not a valid matching, it is considered an

upper boundary to the optimal solution. The primal solution is the result of applying maximum weighted matching on the dual solution. Hence, it is a valid solution for the original problem, but it may be suboptimal and is considered a lower bound to the optimal solution.

Figure 4.5 demonstrates the development of the boundaries over multiple iterations. While the black crosses show the values of the dual and primal solution for each iteration, the red lines mark the yet discovered boundaries. The targeted optimum solution is sketched in blue (of course it is not known a-priori).

Iterative subgradient optimization searches for the maximum of a function by taking successive steps in the direction of a positive gradient. Applied to `LaRA`, in every iteration we adjust each Lagrange multiplier λ_{lm} the following way: If $b_{lm} > b_{ml}$ then λ_{lm} is decreased by a step size s, if $b_{lm} < b_{ml}$ then λ_{lm} is increased by s. For $b_{lm} = b_{ml}$ nothing happens, because constraint 2.7 is satisfied.

This means that in each iteration we push the alignment slightly into the desired direction. The step size s is determined as the bounds distance divided by the number of violations of constraint 2.7: the smaller the bounds distance and the more violations, the smaller becomes the step size.

However, there is a small possibility that the iterations get stuck in a loop of repeating states and as a consequence the bounds would never converge. To spot this, I implemented a counter for the number of non-improving iterations. If there is no improvement of the bounds for 50 iterations, the step size s is halved and the counter is reset, so the algorithm has the chance to break out from the loop.

In the `LaRA 2` implementation I keep track of the lowest upper bound and the highest lower bound values, in order to estimate the remaining distance to the optimum. In addition, I keep track of the best valid alignment, which is updated whenever the best lower bound is increased. In figure 4.5 these are the alignments, which are represented by the black crosses that hit the lower red line.

There are three scenarios for terminating the iterations:

- The dual solution is a valid matching and thus equals the primal solution. In this case we have found the optimum.
- If the distance is smaller than a parameter $\varepsilon > 0$, the solution is close enough to the optimum. The default in `LaRA 2` is $\varepsilon = 0.01$.
- A variable `remainingIterations` is initialized with the maximum number of iterations, which is 500 in the default case. It is decremented in each iteration, and if it reaches zero, the best alignment so far is reported.

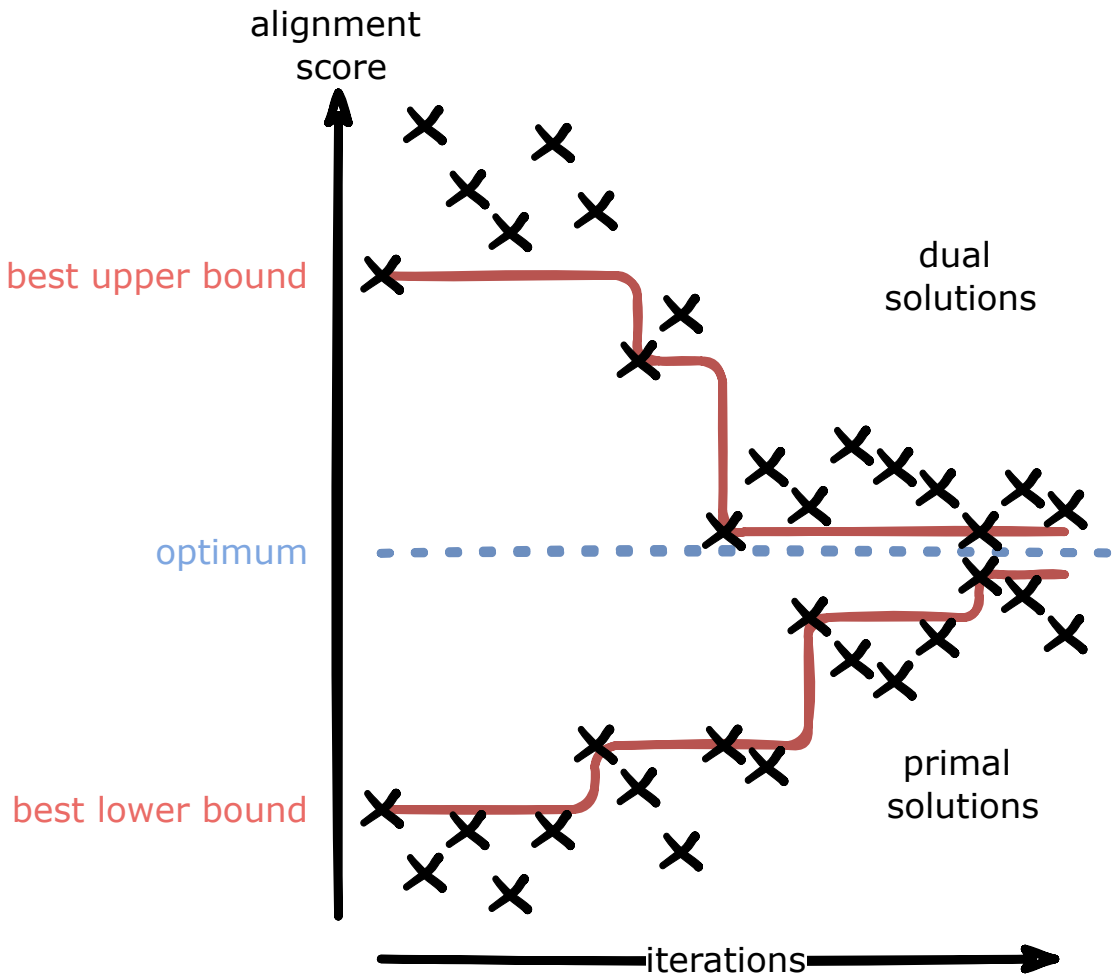


Figure 4.5: **Evolution of the alignments towards the optimum.** The figure visualizes how the best bounds evolve from the dual and primal solutions. The dual solutions are the scores of the relaxed problem, i.e. the alignment step; the primal solutions are the result of the corresponding maximum weighted matching. The more iterations are performed, the closer get the best bounds to the optimum.

5 Parallel and vectorized execution of LaRA 2

One of the most beneficial improvements compared to `LaRA 1` is the possibility of `LaRA 2` to utilize multiple CPU cores at the same time through multithreading and, on top of that, compute multiple alignments per thread through vectorization. The emerging speed-up of up to 130 allows the application of `LaRA 2` on much larger data. In the following sections I describe how I implemented these technologies in `LaRA 2`, together with the challenges I had to face.

A short run time in relation to the problem size is an important aspect. Given the current rapid increase of the size of data sets it is essential to have efficient implementations available that solve the structural alignment problem in reasonable time, while securing a sufficient quality of the results. Some programs already allow to distribute the work on several cores for parallel execution through multithreading. With `LaRA 2` I go a step further and combine multithreading with vectorization: By storing the data of multiple alignments in vectors, they can be computed simultaneously on a single core. Hence, with $\mathcal{T}$ cores and vector size $\mathcal{S}$, `LaRA 2` processes $\mathcal{T} \cdot \mathcal{S}$ alignments simultaneously.

5.1 SIMD vectorization

The term SIMD means single instruction, multiple data. It is a type of parallel processing, where the same operation is performed on multiple data points simultaneously. Instead of e.g. computing the sum of two integer values, SIMD allows us to pass two vectors of integer numbers and a special processor instruction can compute the pairwise sum in a single operation (the result is a new vector, of course).

The vector size $\mathcal{S}$ is system-dependent. There exist different instruction sets, like SSE4, AVX2 and AVX512, which support 128, 256 and 512 bits per vector, respectively. Assuming a data size of 32 bits, there is space for 4, 8, or 16 values in a vector. Note that $\mathcal{S}$ is also dependent on the data size, however in this thesis the data size is considered 32 bits, which is the size of an integer or float.

Previous work has been done on the vectorization of the pairwise alignment computation using the wavefront approach [Daily, 2016, Rahn et al., 2018] and for the recognition of barcode and adapter sequences [Roehr et al., 2017]. My implementation extends the work by Rahn et al., 2018.

A SIMD vector is constructed as `seqan::SimdVector<int>`, which is a data structure in `SeqAn` that provides a system-independent interface for operations on

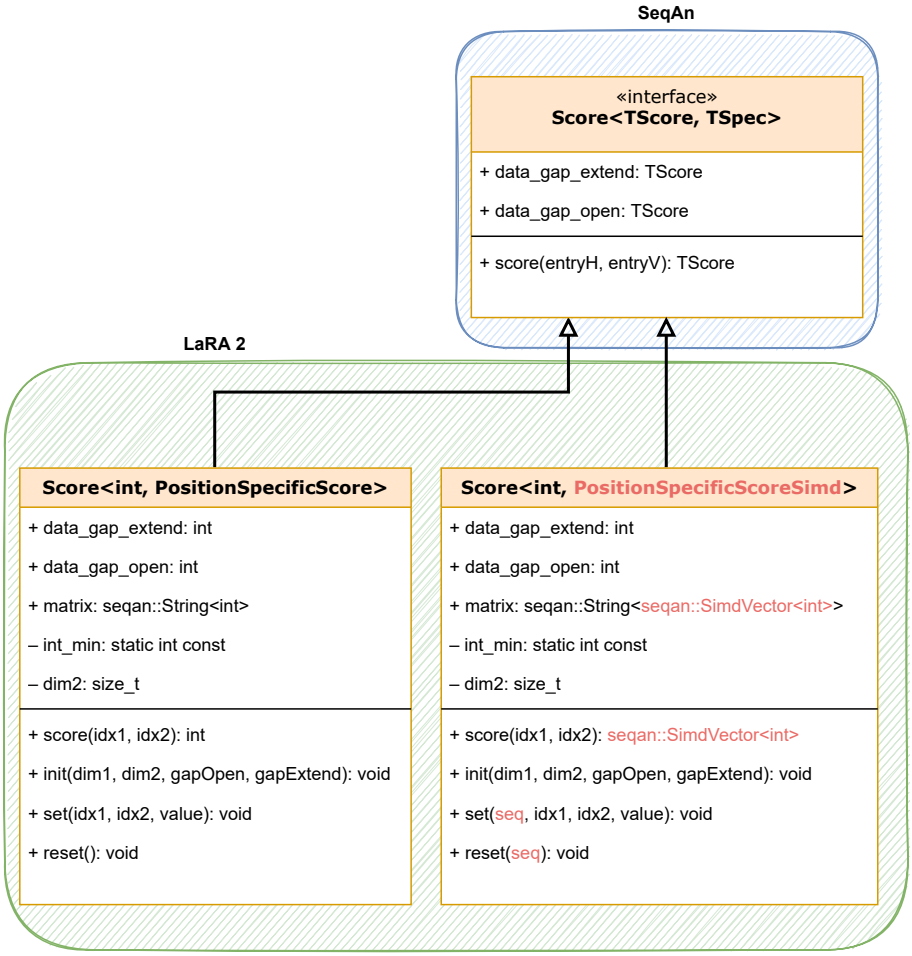


Figure 5.1: **Additional class for position-specific score with SIMD.** The new class uses vectors of score values. Some additional API adaptations to the functions are necessary. The changes to the non-SIMD implementation are highlighted in red.

SIMD vectors and uses internally the Intel compiler intrinsics[1]. This interface automatically applies the vector size that is determined through compilation with one of the instruction sets. During run time, users of `LaRA 2` do not need to care about enabling the SIMD functionality. Instead, this decision is made with the compilation of `LaRA 2`, where the `-march` flag should be used to tell the compiler about the minimal hardware the code should run on, so the instruction set is chosen accordingly. I have described the compiler configuration for clang and gcc in detail in the installation instructions section on the project website and in the readme file.

In section 4.3 I have described a new score class that can cope with position-specific scoring functions rather than a simple character comparison. In order to use vectorization for the structural alignment approach, I implemented a modification

[1] `https://software.intel.com/sites/landingpage/IntrinsicsGuide` (11.07.2022)

```
// Define SIMD vectors for single values, e.g. ones = (1,1,1,1,1,1,1,1)
auto const zeros = seqan::createVector<seqan::SimdVector<int>>(0);
auto const ones = seqan::createVector<seqan::SimdVector<int>>(1);
auto const twos = seqan::createVector<seqan::SimdVector<int>>(2);
auto const maxNonImproving = seqan::createVector<seqan::SimdVector<int>>(50);

while (numAtWork > 0)
{
    // simplified interface to show the idea
    currentUpperBound = computeSIMDalignments(gappedSeq1, gappedSeq2, pssm);
    currentLowerBound = matching(gappedSeq1, gappedSeq2, lagrangeMult);

    // compare upper bound: mask = true where best > current
    auto mask = seqan::cmpGt(bestUpperBound, currentUpperBound);
    // best = mask ? best : current
    bestUpperBound = seqan::blend(bestUpperBound, currentUpperBound, mask);
    // nonImproving = mask ? nonImproving : 0
    nonImproving = seqan::blend(nonImproving, zeros, mask);

    // compare lower bound
    mask = seqan::cmpGt(currentLowerBound, bestLowerBound);
    bestLowerBound = seqan::blend(bestLowerBound, currentLowerBound, mask);
    nonImproving = seqan::blend(nonImproving, zeros, mask);

    // if the limit of non-improving iterations is reached then use half the step size
    nonImproving = nonImproving + ones;
    mask = seqan::cmpGt(nonImproving, maxNonImproving);
    stepFactor = seqan::blend(stepFactor, stepFactor / twos, mask);
    nonImproving = seqan::blend(nonImproving, zeros, mask);

    // decrement remaining iterations
    remainingIterations -= ones;
    // new step size
    stepSize = stepFactor * (bestUpperBound - bestLowerBound) / lagrangeMult.size();

    // [...] update pssm
    // if an alignment is finished: replace it or decrement numAtWork
}
```

Listing 5.1: **Working with SIMD vectors.** This listing shows how the subgradient optimization is implemented with SIMD vectors. It also demonstrates the SeqAn interface that wraps the system-dependent commands, so we do not need to care about the vector size, for instance.

of this score class, named `PositionSpecificScoreSimd`, where each entry of the PSSM is a SIMD vector of length $\mathcal{S}$ instead of a single integer value. See figure 5.1 for a comparison: Further adaptations to the API of this class are the return type of the `score()` function, which returns a SIMD vector, and the `set()` and `reset()` functions, which take an additional sequence index, since we need to update single values inside a SIMD vector.

The presented score class can be employed by the SIMD global sequence alignment algorithm by Rahn et al., 2018, similar to the class for PSSM that we discussed in section 4.3. However, the API of the `globalAlignment()` function cannot be used as is, since it extracts the results from the SIMD vectors in a loop inside. Instead, in `LaRA 2` I still want to proceed with the scores as SIMD vector, so I exclude this step by copying only the few desired code parts of that function to `LaRA 2`.

For invoking the alignment I prepare the pairs of input sequences in the format of two *dependent StringSets* of length $\mathcal{S}$, a slim data structure in `SeqAn 2` that stores only links to the original sequences. The StringSets hold index sequences that have the lengths of the original sequences to be aligned (compare section 4.3). Again, I use the memory-efficient trick to store only the longest index sequence and assign virtual prefixes for the others.

In addition to the alignment step, also the iterative subgradient optimization employs SIMD vectors. The result of the alignment is a SIMD vector of the current upper bound. It is compared to the best upper bound vector, which results in a so-called mask. A mask is a boolean SIMD vector of length $\mathcal{S}$ that contains the result of an operation (in this case the `greater` operation). Afterwards, we can use the `blend` function to set values conditionally based on the mask. In this way I update the best upper and lower bound vectors, count the number of remaining and non-improving iterations, as well as compute the next step size. An excerpt of the corresponding code is shown in listing 5.1.

5.2 Multithreading

Multithreading means to execute independent parts of a program code as threads. On a multi-core system (which are the most of the systems nowadays), threads can run simultaneously on different cores and thus execute the program *in parallel*. Optimally, we try to distribute work equally to the cores of the system.

The optimal number of threads is system-dependent and in `LaRA 2` it is determined at run time, according to the following priority:

1. the value of the `--threads` parameter
2. the hint returned by `std::thread::hardware_concurrency()`
3. a single thread if the previous methods both return 0

The parallel execution of `LaRA 2` with two threads on an exemplary dual-core system with AVX2 instruction set ($\mathcal{S} = 8$) is visualized in figure 5.2. It shows the independent execution of the two threads: each thread performs the `LaRA` iterations of alignment and maximum weighted matching on its individual set of 8 alignments. The SIMD bound computations and PSSM update are also performed in the thread, but since they are so fast compared to the other steps, they are not shown.

As discussed at end of section 4.3, the list of alignments to be computed is sorted in descending order by the length of the longer sequence. A key advantage of this order is clearly visible here: The alignments that are grouped together in a SIMD vector have similar length. Since the run time of a SIMD alignment is determined by the longest contained alignment, the second (shorter) group can iterate significantly faster than the first group.

I have put much effort into trying to parallelize also the step that computes the structures with `RNAlib`. However, the library version that was available at the time

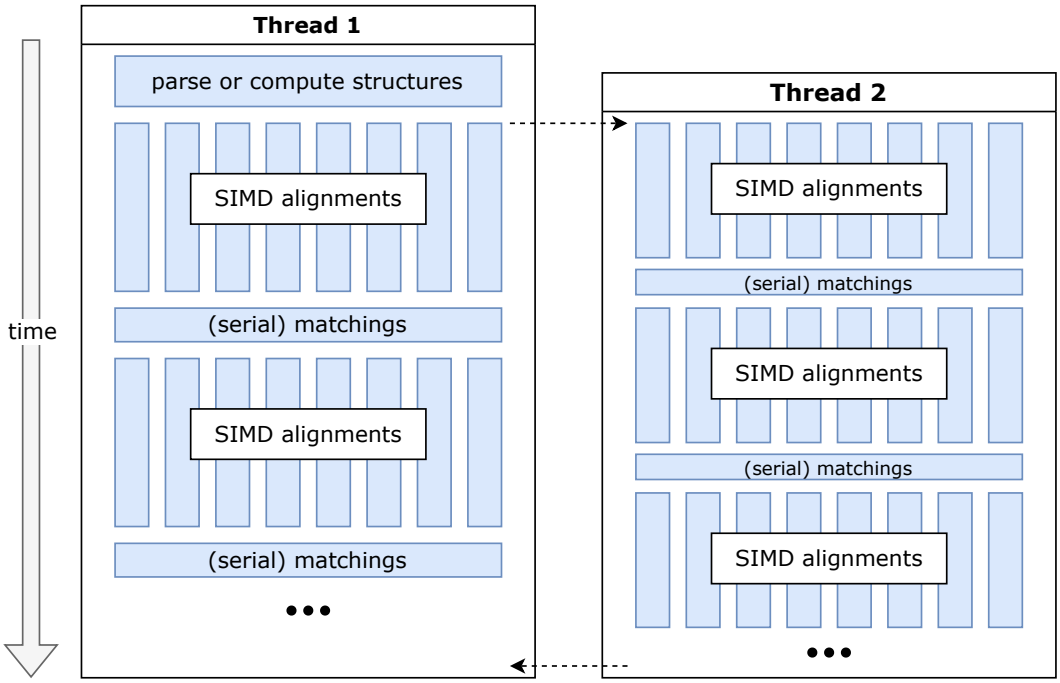


Figure 5.2: **Execution flow with multiple threads.** Each thread solves $\mathcal{S}$ different alignments at the same time. The figure shows the benefit of sorting the alignments by length: thread 2 has a group of shorter alignments and can iterate faster. The threads are mostly independent; only if an alignment is finished and has to be replaced, a shared access is needed.

I was implementing `LaRA 2` turned out to be not thread-safe. For more information I refer to the last paragraph of section 3.3.

If the iterations for one of the alignments in a SIMD group should terminate (for one of the three scenarios described on page 43), the result of that specific alignment is stored and the next alignment is constructed in-place. Since multiple threads can finish an alignment at the same time, two mutexes are used to lock (1) the iterator to the next sequence pair and (2) the next free storage for finished alignments. The function of a mutex is to allow only a single thread at a time the access to a shared resource.

At the end of this section, I briefly want to sketch a concept that I had to discard. In an earlier development step of `LaRA 2`, the command flow had the out-most loop over the iterations. In each iteration we would compute in parallel the SIMD alignments of all given sequence pairs, followed by all the respective matchings. This implementation works for a small set of sequences, but with increasing amount of sequences n the number of alignments grows quadratically to $\frac{1}{2}n(n-1)$. Keeping all these alignments in memory has turned out to be problematic, since each alignment needs storage for the position-specific score matrix, the base pair probability matrix, the graph for MWM, the bounds values, and further individual values for the subgradient optimization (e.g. current step size, number of non-improving iterations).

Because of the much lower memory footprint, I decided to compute at most $\mathcal{T} \cdot \mathcal{S}$ alignments at a time. A second argument against looping the iterations is that each alignment requires an individual amount of iterations, which is not known a-priori. Thus, the algorithm would have to run the maximum number of iterations for all the alignments, although the most of them would idle after their optimum has been found.

5.3 Testing the speed-up

In this section I want to demonstrate the scaling of the run time of pairwise structural alignments with `LaRA 2` in the light of SIMD instruction sets and multi-threading. As a benchmark I use the plastids data set from the 5SrRNAdb [Szymanski et al., 2002] database, which contains 838 sequences with average length 123. This results in 350,703 pairwise structural alignments. Figure 5.3 visualizes the run time for computing all these pairwise alignments with `LaRA 2`, including 25 seconds for the non-parallel base pair probability calculations. For this test, `LaRA 2` was compiled with gcc version 9 on a Linux server with 126 GB RAM and an Intel® Xeon® CPU E5-2650 v3 with 2.30 GHz.

The effect of SIMD instructions is a speed-up of 1.8×–1.6× with AVX2 and 1.6× with SSE4. Because the vectorization is implemented for the alignment step and not for the matching and folding, these factors are reasonable. In combination with multi-threading there is a large improvement of the run time. With 16 threads `LaRA 2` achieves 13× speed-up compared to the single-threaded run in the SSE4 or non-SIMD case and 11.5× with AVX2. The sequential part in the program is mainly the computation of the base pair probabilities with `RNAfold`, which takes constantly 25 seconds. A limiting effect to the speed-up has a larger memory allocation, e.g. for AVX2 instructions and 16 threads the program needs to allocate 128 alignments.

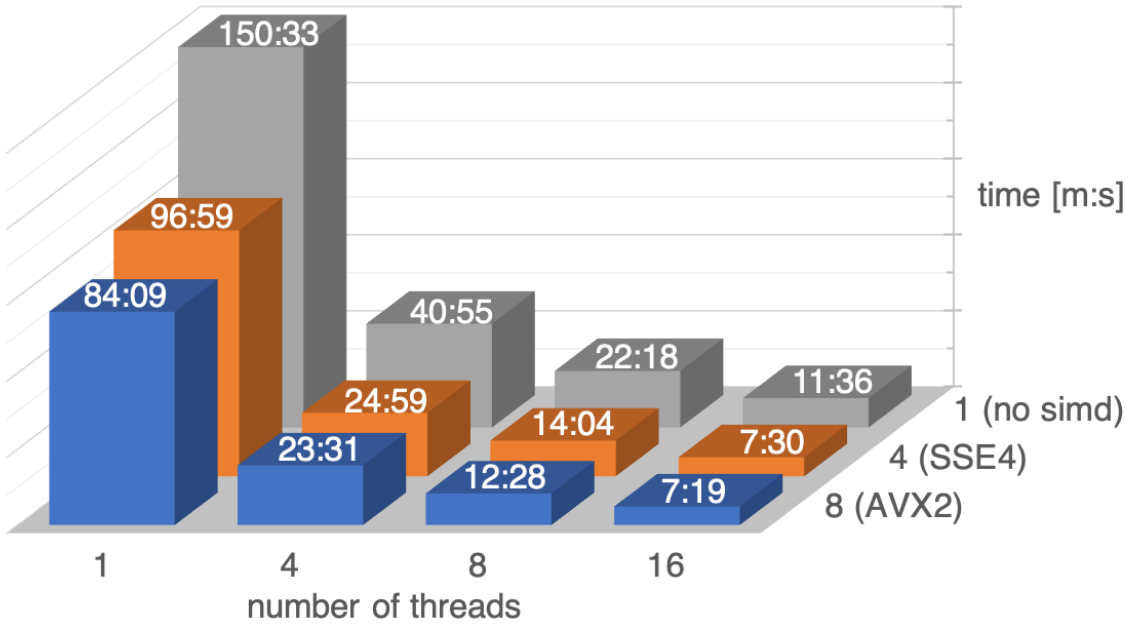


Figure 5.3: **Run time comparison for various thread and SIMD configurations.** In the bar-graph, the time for computing 350,703 pairwise alignments is reported in minute:second [m:s] notation. The matrix shows the run time for 1, 4, 8 and 16 threads with different SIMD instruction sets, which compute 1, 4 or 8 alignments per thread.

6 Output and multiple alignment

After all the pairwise alignments are computed, LaRA 2 produces one of the following three output formats, which can be selected with the `--outformat` parameter:

1. MSA library for `T-Coffee` [Notredame et al., 2000]
2. pairwise alignments for `MAFFT` [Katoh and Toh, 2008]
3. aligned *Fasta* format (only if exactly two sequences are given)

I have implemented all the output functionality in a class named `OutputLibrary`. This class is responsible for collecting the finished pairwise alignments and printing the results in one of the three formats to a file or to stdout.

The output library is the data structure that is visualized in figure 6.1. It stores a reference to the sequence records, because it needs to access the sequences and sequence names for writing the output. The set of optimal alignments is stored as a pair of the actual alignment data structure and its score. The alignment data structure consists of a pair of sequence indices and a vector containing triples of all the base pairings with their individual scores. These scores correspond to the sequence and structure conservation in the associated nucleotide pair. Lastly, the output library stores the selected output format.

A structural multiple alignment (MSA) is an alignment of more than two sequences. As introduced in the beginning of chapter 4, the pairwise alignments from LaRA 2 need to be processed by a progressive alignment method afterwards that combines them to a multiple alignment. The following sections describe each of the three output formats in detail and how they can be further processed.

6.1 T-Coffee

The introduced alignment data structure is designed to print an MSA library for `T-Coffee` [Notredame et al., 2000] easily and without further calculations. `T-Coffee` is a well-known and fast software for progressive multiple sequence alignment. It incorporates structural information by constructing an alignment graph that contains the structural weights of the pairwise alignments.

The library data structure consists of a weighted set of sequences with weighted character pairings. The major advantage of `T-Coffee` over other tools is that due to its library design it is flexible enough to support also the incorporation of other constraints (from e.g. already computed alignments) or additional, experimentally

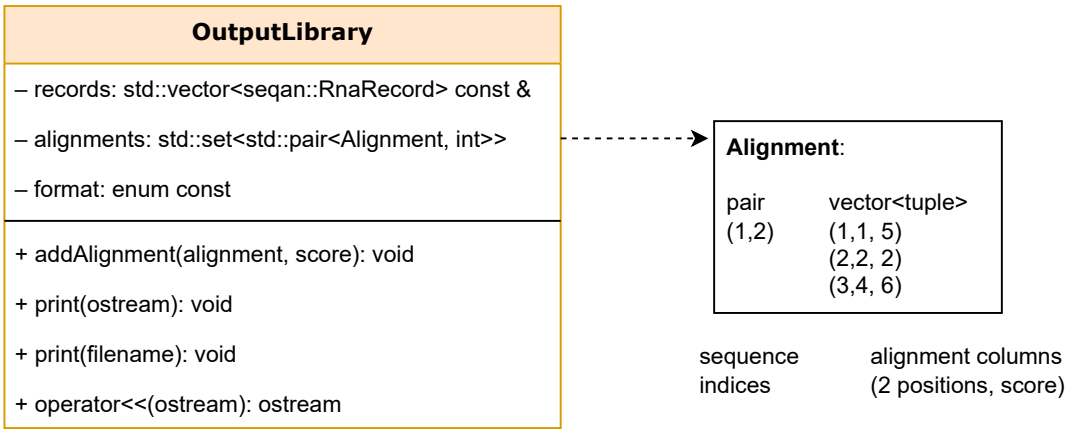


Figure 6.1: **Class diagram for the output library.** It shows the functions for collecting and printing the alignments, as well as the member variables that store the necessary data. The alignment data structure is sketched on the right-hand side and consists of indices for the associated sequence pair, as well as a vector of triples that contain the start position, end position, and score for each alignment line. For instance, the last entry shown can be interpreted as follows: Align base 3 of sequence 1 with base 4 of sequence 2, and the score contribution is 6. This implicitly means that base 3 of sequence 2 is aligned to a gap character.

gained structures (e.g. obtained by SHAPE experiments) by adjusting the weights accordingly in the library. These can be adjusted right in the library file or by using the software's input options.

`SeqAn 2` has an implementation of the `T-Coffee` algorithm that is in many aspects superior to the original program. Instead of observing each nucleotide individually, `SeqAn::T-Coffee` by Rausch et al., 2008 applies so-called segments, which consist multiple adjacent nucleotides. This leads to a segment graph that is much smaller than an alignment graph and is thus much faster, especially for long sequences. Further improvements for deep alignments have been implemented by Yasnev, 2015. Together with Svenja Mehringer I have already developed the architecture for a `SeqAn 3`-based version of `T-Coffee`, but it is not subject to this thesis.

Unfortunately, `SeqAn::T-Coffee` does not work as expected with `LaRA 2`. The results have always been much worse in the benchmarks compared to the original program by Notredame et al., 2000. I have tried various older versions from the repository and also excluded the code that was added for deep alignments, but in the end I came to the conclusion that the segment approach messes up the structural information included in the pairwise alignments and that the original tool should be used.

`T-Coffee` can be invoked simply with `t_coffee -lib results.lib`, where the file name should be the same that has been used as output parameter (`-w`) for `LaRA 2`. The format of this file is shown in listing 6.1: It in enclosed in a header and footer

```
1  ! T-COFFEE_LIB_FORMAT_01
2  3
3  seq1 6 AAACCC
4  seq2 6 CCCGGG
5  seq3 6 AAACCC
6  # 1 2
7  4 1 1000
8  5 2 1000
9  6 3 1000
10 # 1 3
11 1 1 1000
12 2 2 1000
13 3 3 1000
14 4 4 1000
15 5 5 1000
16 6 6 1000
17 # 2 3
18 1 4 1000
19 2 5 1000
20 3 6 1000
21 ! SEQ_1_TO_N
```

Listing 6.1: **Example of the MSA library format.** Each record describes an alignment and consists of a block that starts with `#` and two sequence indices (lines 6, 10, 17). The following lines until the next block are triples with two nucleotide positions and a score, as seen before in figure 6.1. Note that the indices for sequences and nucleotides are counted from 1 in this file format. In line 2 there is the number of sequences, followed by a list of the sequences' names, length and the nucleotides.

line starting with `!`, contains the number and lengths of all the sequences with their description on top, and is followed by records with the alignment information as indicated above. From the format it is obvious that further lines with additional constraints can be added easily.

6.2 MAFFT X-INS-i

The pairwise alignment output of `LaRA 2` is designed to be parsed by the `MAFFT` framework [Katoh et al., 2002] and contains three lines per pairwise alignment, as shown in listing 6.2. The first line is a header line similar to the *Fasta* format containing both sequence identifiers, and the remaining two lines consist of the first and the second aligned sequence. The aligned sequences possibly contain gap symbols and have equal length.

`MAFFT` is well-known as a fast multiple sequence aligner, which (in its default mode) does not care about secondary structure. However, there is an extension to the `MAFFT` framework called `X-INS-i` [Katoh and Toh, 2008] that performs a multiple sequence-structure alignment. As a first step, it uses the algorithm by McCaskill, 1990, which we have discussed in section 2.1.3, in order to compute the base pair probabilities of the input sequences. Afterwards, there is the most interesting step:

```
>seq1 && seq2
AAACCC---
---CCCGGG
>seq1 && seq3
AAACCC
AAACCC
```

Listing 6.2: **Pairwise alignment file for MAFFT.** This format looks similar to *Fasta*, but it contains two aligned sequences per record.

```
// old
sprintf( com, "env PATH=%s:/bin:/usr/bin mafft_lara -i _larain -w _laraout -o _lara.params %s",
        whereispairalign, laraarg );
// new
sprintf( com, "env PATH=%s:/bin:/usr/bin mafft_lara -i _larain -w _laraout -o pairs %s",
        whereispairalign, laraarg );
```

Listing 6.3: **Amendment of MAFFT X-INS-i to use LaRA 2.** Only a single option needs to be adapted in file `core/pairlocalalign.c`. It sets the desired output format with `-o pairs`.

it computes pairwise structural alignments with `LaRA 1` [Bauer and Klau, 2005] or `SCARNA` [Tabei et al., 2006].

It was easy to amend the `X-INS-i` workflow such that it is able to use `LaRA 2`: There is one code line that builds the command line call to `LaRA`, and instead of passing a parameter file (this was the way to set options for `LaRA 1`), I set the option of `LaRA 2` that creates the pairwise alignment output that is expected by `X-INS-i`. The exact code change is defined in listing 6.3.

The fork repository of `MAFFT` that contains the change is located at GitHub and the command for running `X-INS-i` with `LaRA 2` is `mafft-xinsi --larapair input.fasta`.

The workflow of `X-INS-i` computes a progressive MSA based on a guide tree (like `T-Coffee`), but applies an iterative refinement step with a so-called four-way-consistency score [Katoh and Toh, 2008]. It incorporates the structural information not only through the (here unweighted) base pairs of the pairwise alignments, but additionally from the initial base pair probabilities resulting from the McCaskill algorithm. The four-way-consistency score for an interaction match (defined in section 2.2.2) is derived from the involved based pair probabilities and a sequence similarity score obtained from the algorithm by Vingron and Argos, 1990.

6.3 Pairwise alignment output

In case of $n = 2$ there is no need for generating a multiple alignment, because there is only one pairwise alignment. Since it is not necessary to invoke another tool in this case, I have implemented an aligned *Fasta* output for two sequences. This format is accepted by most existing tools that take an alignment as input, e.g.

```
>seq1
AAACCC---
>seq2
---CCCGGG
```

Listing 6.4: **The Fasta alignment file format.** This format can represent two or more aligned sequences with gap characters. All sequences are required to have equal length.

`Biopython` [Cock et al., 2009]. Also `T-Coffee` and `MAFFT` support this format for writing a multiple alignment.

Like in a *Fasta* sequence file, there are two lines per sequence recorded: an identifier and the sequence (here with gap symbols). An example can be inspected in listing 6.4.

7 Benchmarks

In order to demonstrate the performance of LaRA 2 compared to relevant existing software, I have evaluated three different benchmarks with focus on multiple alignment with conserved structures, run time comparison on a large data set, and the detection of pseudoknots.

All benchmarks have been performed on a Linux server using an x86_64 architecture with Intel® Xeon® CPU E5-2650 v3 with 2.30 GHz and 126 GB RAM. I compiled with GCC version 9 and where applicable, I used up to 16 threads and AVX2 instructions.

For the benchmarks I have chosen the best tools from recent publications, as analysed in section 2.2.

The employed program versions and parameters are displayed in table 7.1 and include LaRA 2 [Winkler et al., 2022] with both MAFFT [Katoh et al., 2002] and T-Coffee [Notredame et al., 2000] as multiple alignment, LaRA 1 [Bauer and Klau, 2005] linked with T-Coffee version 5.0.5, the structural aligner LocARNA [Lorenz et al., 2011], and RNAmountAlign [Bayegan and Clote, 2020].

In addition, the standalone MAFFT tool [Katoh et al., 2002] is included as a pure sequence aligner in order to demonstrate that structural alignment is superior for ncRNA comparisons.

For the purpose of a fair comparison with other tools I include the time for folding the sequences in the benchmarks (unless stated otherwise). However, if the structure annotation is available the folding step can be omitted for LaRA 2.

LaRA 2	v2.0.1	`lara --threads 16`
+ MAFFT	v7.453	`mafft-xinsi --larapair`
+ T-Coffee	v13.41.0	`t_coffee`
LaRA	v1.4.3	`lara`
LocARNA	v1.9.0	`mlocarna --threads=16`
RNAmountAlign	v1.0	`RNAmountAlign`
MAFFT	v7.453	`mafft --thread 16`

Table 7.1: **Program versions and parameters for the benchmark.** If possible and not already default, I chose 16 threads for execution, and used the most recent program versions as of April 2021.

7.1 Multiple alignment of RNA families

This benchmark compares the performance for multiple structural alignment across several RNA families dependent on the sequence similarity. I took the BRAliBase 2.1 data set, which consists of 388 reference alignments of 5 sequences each (after removal of the 93 SRP alignments due to an erratum). The contained RNA families are 5S ribosomal RNA (RF00001), tRNA (RF00005), U5 spliceosomal RNA (RF00020), and Group II catalytic introns (RF00029) [Gardner et al., 2005]. For the evaluation of `RNAmountAlign` I had to exclude 46 alignments that contain the character 'N', because the program does not accept wildcard symbols.

In order to evaluate the resulting multiple structure alignments two metrics are applied: SPS and MCC. The sum-of-pairs (SPS) score is a measure of similarity between the test alignment and the curated reference alignment that is available in the Rfam database [Kalvari et al., 2018]. SPS values are in range [0..1], where 1 means identity and value 0 represents maximal distance. While SPS considers solely the character matchings, the Matthews correlation coefficient (MCC) [Matthews, 1975] evaluates the predicted secondary structure. MCC is a value in range $[-1..1]$, where 1 denotes a perfect prediction, 0 is a random prediction according to the background distribution, and -1 denotes a total disagreement.

For calculating the MCC as shown in equations (7.1) and (7.2), I follow the publications of the tools `Murlet` [Kiryu et al., 2007] and `RNAmountAlign` [Bayegan and Clote, 2020]. For future reference and reproducibility the script is provided in the `LaRA 2` repository.

In a first step the test alignments are folded with *RNAalifold* from the `ViennaRNA` package [Lorenz et al., 2011]. I have computed the consensus structures with `PETfold` [Seemann et al., 2011] as well, which led to the same results. For the reference alignments the structure annotations from the Rfam 5.0 database with accession numbers RF00001, RF00005, RF00020, and RF00029 are used, this is according to the compilation of the BRAliBase data [Gardner et al., 2005].

In the next step the consensus structure is assigned to each sequence of the respective alignment. For all matching base pairs the sequence positions are extracted per sequence and stored in two sets: T_x contains the base pairs of sequence x in the test alignment and R_x contains the base pairs of x in the reference alignment. Based on these sets the amount of true positives (tp), false positives (fp), false negatives (fn), and true negatives tn, which is the so-called confusion matrix, are calculated according to equation (7.1). Note that the true negative (tn) value contains the number of all possible base pairs that are contained in neither the test nor the reference set (and thus it is typically very large). The values of the confusion matrix in turn are used in equation (7.2) to calculate an MCC value.

$$\begin{aligned} tp &:= \sum_x |T_x \cap R_x| & fp &:= \sum_x |T_x \setminus R_x| \\ tn &:= \sum_x |(T_x \cup R_x)^c| & fn &:= \sum_x |R_x \setminus T_x| \end{aligned} \tag{7.1}$$

$$\text{MCC} := \frac{tp \cdot tn - fp \cdot fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}} \tag{7.2}$$

Figure 7.1 shows the performance of the tested tools according to the SPS and MCC benchmarks. The curves in a) and b) are fitted through the data points with a lowess smoother ($f = 0.5$). The statistical significance of the MCC benchmark is displayed in c). As annotated in the BRAliBase data set [Gardner et al., 2005], I have divided the alignments in three groups of low, medium and high sequence similarity. For each group and each tool I have calculated the median and 95% confidence intervals after bootstrapping 1000 samples.

The results demonstrate that `LaRA 2` performs as good as `LocARNA` and `LaRA 1`, and better than `RNAmountAlign` and `MAFFT`. In the alignments with more than 70% sequence similarity we observe the same performance for all tools in the SPS benchmark. This is expected, as the importance of the structure is low and even a pure sequence aligner is able to compute alignments that are close to the reference alignment. For lower sequence similarities we observe an almost linear regression in the SPS score of `MAFFT`, because the structure becomes more crucial. Here we observe that `LaRA` and `LocARNA` clearly perform the best among the tested tools.

Another question that has concerned me is the performance drop of all programs around the 55% sequence similarity region in the SPS benchmark. As Löwes et al., 2016 have pointed out, this is the effect of an unbalanced representation of RNA families in the BRAliBase benchmark set: Especially the tRNA family, which has a well-conserved cloverleaf structure, is overrepresented in the BRAliBase data set and bears responsibility for the dent.

For the structure evaluation with Matthews correlation coefficient, `LaRA 2` has the same performance as `LocARNA` and `LaRA 1`, while this group outperforms `MAFFT` and `RNAmountAlign`. An interesting observation is the decline of the reference curve for high sequence similarity, which is mainly represented by alignments of the tRNA family. For the reference curve I computed the optimal structures of the BRAliBase reference alignments with `RNAalifold` [Lorenz et al., 2011] (they do not provide reference structures), and compared them with the respective curated structures from Rfam, using the MCC metric. Apparently, the results of all the programs follow the same trend as the reference, while for higher sequence similarity the curves get closer to each other.

I was surprised to see that above 55% sequence similarity `MAFFT` has a better performance than `RNAmountAlign` in the MCC benchmark, as shown in figure 7.1 b) and c). The comparably poor performance of `RNAmountAlign` for low sequence similarities is compliant with the results that have already been published by Bayegan and Clote, 2020. My assumption is that `RNAmountAlign` balances the weight too much on the sequence similarity.

The run time for the benchmark is displayed in figure 7.2. I summed up the run time for 481 executions of each tool (including the SRP data in order to have more

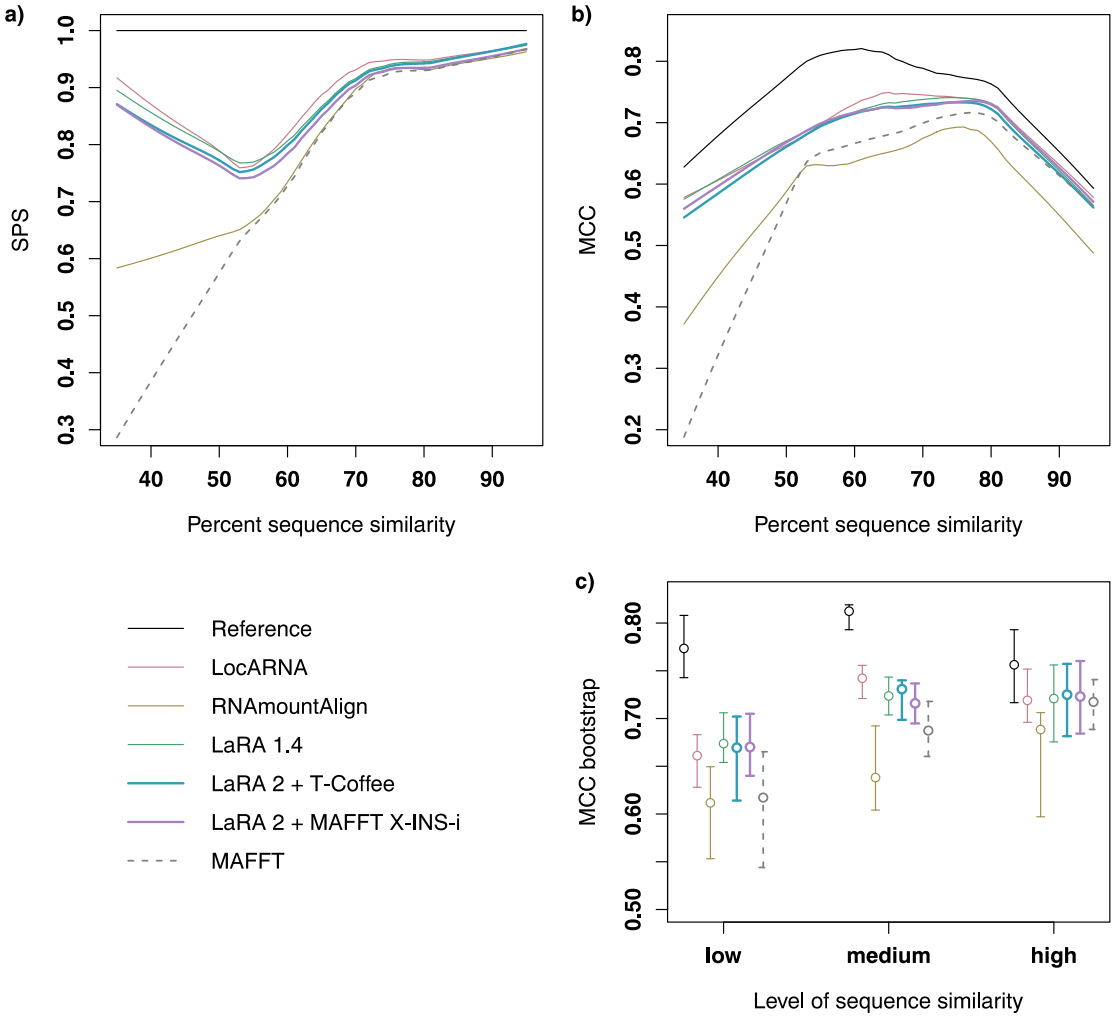


Figure 7.1: **SPS and MCC evaluation for the BRAliBase data set.** a) Sum-of-pairs score and b) Matthews correlation coefficient are shown for different tools dependent on the sequence similarity. The tools were run on 388 alignments of the BRAliBase 2.1 data set (without SRP) and the curves were generated with a lowess function on the results. In order to show the MCC performance of MAFFT as a sequence alignment tool, as well as of the reference alignment from BRAliBase, we calculated the best secondary structure of the alignments with `RNAalifold` [Lorenz et al., 2011]. c) 95% bootstrap percentile confidence intervals and medians for the MCC values. The first axis represents the sequence similarity in three groups: low ($< 55\%$), medium ($\geq 55\%$ and $< 75\%$) and high ($\geq 75\%$), as annotated in BRAliBase [Gardner et al., 2005]. For each group we bootstrapped 1000 samples of the MCC experiment.

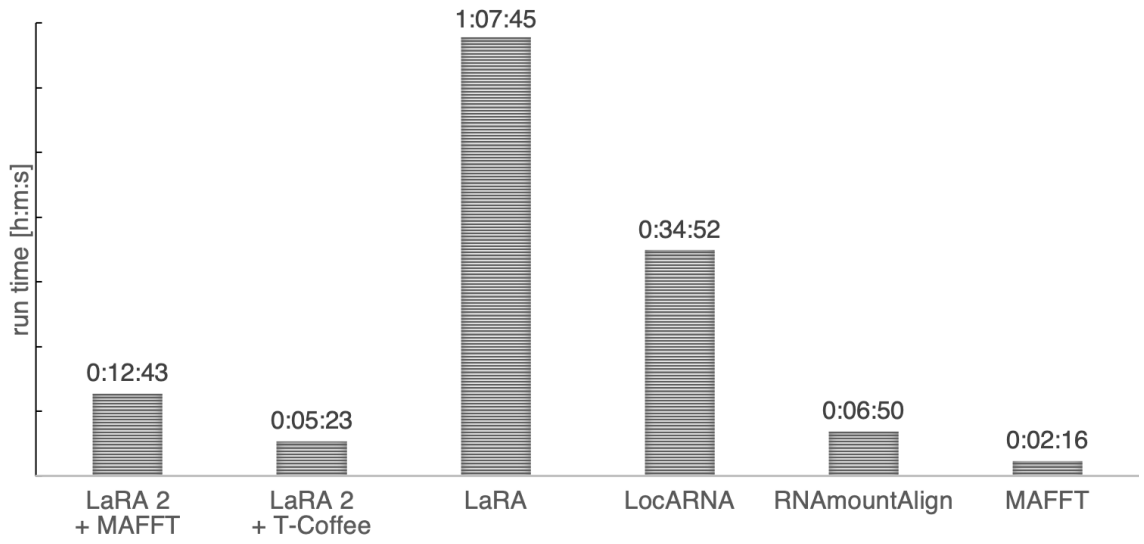


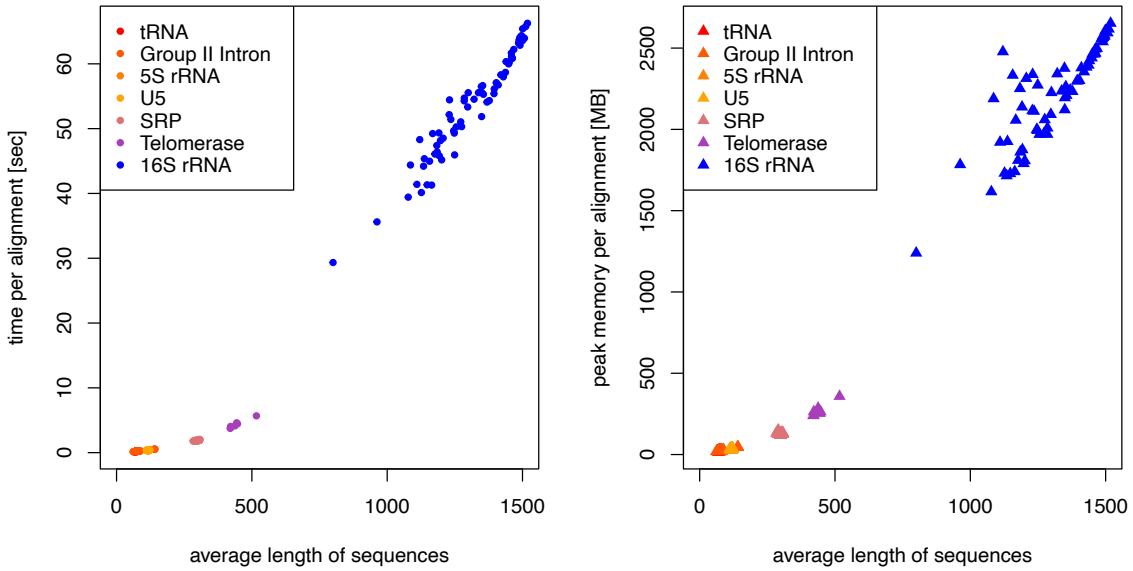
Figure 7.2: **Run time of the tested programs for 481 alignments** of 5 sequences each from the BRAliBase 2.1 benchmark, including the SRP data. The calculation of the base pair probabilities is included in the run time. The displayed run time of `RNAmountAlign` is scaled with factor $\frac{481}{384}$ for comparison.

data points), except `RNAmountAlign`, which I ran on a limited set (384 alignments that do not contain wildcard characters) and scaled the run time accordingly.

The fastest result of the sequence-structure aligners is delivered by `LaRA 2` with `T-Coffee` in less than 5.5 minutes. This is closely followed by `RNAmountAlign` (below 7 minutes), which is impressive in the light of its non-parallel execution, but shadowed by its performance in the benchmark above. `LaRA 2` with `MAFFT` runs in less than 13 minutes, `LocARNA` takes almost 35 minutes and the single-threaded `LaRA` requires more than 1 hour to compute the test alignments. `MAFFT` is the fastest among all tested tools, however this is expected since sequence alignment is a less complex problem. As there are only five sequences aligned at a time, parallel execution has just a minor effect compared to the benchmark in section 7.3.

7.2 Influence of the sequence length

I examined the time and memory consumption of `LaRA 2` with respect to the average sequence length. As the BRAliBase data set contains rather short sequences (up to 300 bases) I extended the set with two additional RNA families from the RNAStrAlign database [Tan et al., 2017]: Telomerase and 16S rRNA. Each alignment consists of five sequences, and I averaged the run time per alignment over 10 runs in order to gain more accurate results. Figure 7.3a shows the results for the run time and figure 7.3b for the maximum allocated memory. In both cases we observe a monotonic increase with sequence length, and an alignment of average sequence length 1500 takes about



(a) Run time in relation to sequence length. (b) Memory in relation to sequence length.

Figure 7.3: **Run time and memory of LaRA 2 in relation to the sequence length.** I used the sequences from BRAliBase 2.1 including SRP as well as Telomerase RNA and Mollicutes' 16S rRNA from RNAStrAlign database [Tan et al., 2017]. Each of the 560 alignments consists of 5 sequences, of which the average length is denoted on the x-axis. The y-axis shows the run time or peak memory consumption respectively for each alignment computation, including the calculation of the base pair probabilities.

	# threads	LaRA 2 + MAFFT	LaRA 2 + T-Coffee	LaRA	LocARNA	RNA-mount-Align	MAFFT
Time [m:s]	16	26:28	54:14	na	419:59	na	0:04
	1	237:00	151:17	3424:57	1260:50	212:30	0:02
Mem [MB]	16	2059	3917	na	3003	na	357
	1	1362	3908	4172	453	3923	36

Table 7.2: **Run time and memory consumption** for the computation of a multiple alignment with 838 sequences of 5S rRNA Plastids, taken from the 5SrRNAdb database [Szymanski et al., 2002]. The comparison shows the tested programs employing 1 or 16 threads. The calculation of the base pair probabilities is included.

one minute and occupies at most 2.6 gigabytes of memory. This benchmark does not include MSA calculations, but includes the computation of base pair probabilities.

7.3 Deep alignments

With deep alignment, I mean an alignment that consists of many sequences. In order to demonstrate the ability of LaRA 2 to process such large data sets in reasonable time, I use the plastids data set from the 5SrRNAdb [Szymanski et al., 2002] database, which contains 838 sequences with average length 123. This results in 350,703 pairwise structural alignments that are then combined to a single multiple alignment. As table 7.2 demonstrates, LaRA 2 with MAFFT X-INS-i can compute this in 26.5 minutes due to its efficient and parallel implementation. The run time of LaRA 2 with T-Coffee is about 54 minutes, and I examined that in both cases the common pairwise alignment part takes less than 7.5 minutes.

MAFFT is significantly faster in this benchmark due to the fact that it is a pure sequence aligner, which is a less complex problem. Interestingly, the multi-threaded version is even disadvantageous for MAFFT, likely because of the larger memory allocation. As stated in section 2.2, LocARNA has a worse run time complexity compared to LaRA 2, which leads to a significantly slower execution with this large alignment. Note that RNAmountAlign and LaRA support only single-threaded execution. I computed also the SPS scores for the results of this benchmark, which are values between 0.95 and 0.98 for all programs.

The speed-up of LaRA 2 with MAFFT using 16 threads is about 9, which is much better than all the other tools. With T-Coffee it reduces to a factor of 3. Still this is the same speed-up factor as LocARNA.

Taking a look at the peak memory allocation in table 7.2 reveals that even with so many sequences the calculations do not require an extensive amount of memory. The maximum allocation of around 4 gigabytes is reached when running RNAmountAlign or T-Coffee (after LaRA 2 or LaRA 1). When LaRA 2 is executed with MAFFT X-INS-i

RNA family id	RF01089	RF01084	RF00499	RF00165
sequence similarity	59.46%	53.51%	81.67%	66.92%
LaRA 2 + T-Coffee	0.82	**0.79**	0.93	0.84
LaRA 2 + MAFFT	**0.86**	0.75	**0.94**	**0.94**
LaRA	0.81	0.77	0.91	0.83
LocARNA	0.76	0.68	0.89	0.80
RNAmountAlign	0.63	0.67	0.93	0.83
MAFFT	0.70	0.59	0.92	0.83

Table 7.3: **SPS evaluation on pseudoknotted structures from Rfam.** The best values are printed in bold font. For each RNA family the sequence similarity is also provided, since it has an impact of the performance of some tools.

the peak memory is determined by the LaRA 2 part, which is around 2 gigabytes for 16 threads and 1.3 gigabytes for single threaded execution. Since this is lower than any other program in multi-threaded mode, I recommend using LaRA 2 with MAFFT if the available memory is limited.

7.4 RNA structures with pseudoknots

Although Danaee et al., 2018 have estimated that 12% of RNA structures contain at least one pseudoknot, the most structural alignment methods do not implement mechanisms to conserve pseudoknotted structures, because their detection is computationally more demanding. Since many commonly used software tools do not detect pseudoknots, the number 12% may still be underestimated. Generally, in alignments with a high enough sequence conservation a pseudoknot can be aligned correctly by any method that aligns for sequence similarity, while for alignments with low sequence similarity the ability of the methods to represent crossing structures becomes more important.

I show with SPS values of some pseudoknotted RNAs from Rfam and in an additional graphical example that LaRA 2 actually detects pseudoknots. SPS scores express the similarity to the reference alignment and therefore a high score indicates that the pseudoknot is aligned properly, however a low score can result from a different location and is not sufficient to prove the absence of the pseudoknot in the test alignment.

The scores in table 7.3 show that LaRA 2 performs the best according to the SPS criterion. This is expected, because LaRA 2 and LaRA receive their structural information from individual base pair probabilities and can model pseudoknots in their graph representation. The high scores of the structural interactions of the pseudoknot benefit the conservation of the respective columns of the multiple

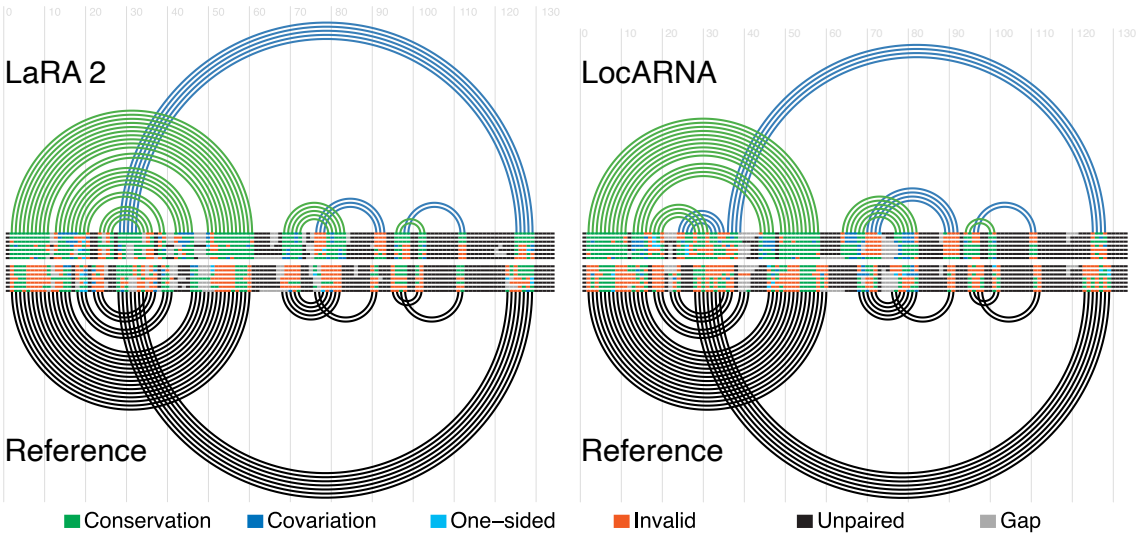


Figure 7.4: **Structure plot of RF01089** with `LaRA 2` and `LocARNA` (north) as well as the reference (south). It was created with `R-chie` [Lai et al., 2012] of type double covariance plot after structure prediction with `IPKnot` [Sato et al., 2011]. The reference result is shown in black, the `LaRA 2` and `LocARNA` results in green/blue. The colour coding of the alignment: If a sequence can form a base pair as dictated by the structure, the base pair is coloured green, else red. For green base pairs, if a mutation has occurred, but base pairing potential is retained, it is coloured in blue (dark for mutations in both bases, light for single-sided mutation). Unpaired bases are displayed in black and gaps in grey.

alignment as shown in the example above. In contrast, a pure sequence aligner like MAFFT can only show good results with high sequence similarity like RF00499.

I have chosen a structure for the graphical example where the pseudoknot interactions are biologically essential and where the alignment is not obvious through sequence comparison to point out the benefit of detecting pseudoknots. Athanasopoulos et al., 1999 describe a pseudoknot in the regulatory region of the repBA gene, which consists of two complementary sequences of 8 bases. The base pairing between them forms a pseudoknot that is essential for translation. I have downloaded for this benchmark the respective seven seed sequences (accession RF01089) from the Rfam database [Kalvari et al., 2018] as well as the respective reference alignment.

I have computed the structural multiple alignment with `LaRA 2` and `LocARNA`. From these alignments I ran the tool `IPknot` [Sato et al., 2011] (mode: McCaskill model with refinement, allow pseudoknots) to produce a secondary structure of the two alignments and the reference in order to detect whether the alignments have the correct pseudoknot positions aligned. Figure 7.4 visualizes the secondary structure from `IPknot` with the `R-chie` tool [Lai et al., 2012]. The southward arcs (black) visualize the interactions in the reference, which can be directly compared with `LaRA 2` and `LocARNA` in the north half (green/blue) of the plots.

The pseudoknot of subject is the long-range interaction that is displayed as large blue arcs in figure 7.4. Comparing the two plots reveals that `LaRA 2` correctly aligned the pseudoknot (mostly green coloured base pairs) and placed it in the same position as in the reference; while with `LocARNA` the left side of the pseudoknot cannot be correctly spotted and is thus not represented in the alignment. In numbers, the reference structure has 39 interactions, of which 11 are in a pseudoknot. Of those, `LaRA 2` correctly detects 32 interactions, including 9 of the pseudoknot ones; `LocARNA` detects only 18 interactions, including 2 pseudoknot interactions.

8 Discussion

The first implementation of the T-LaRA algorithm by Bauer et al., 2007 computes sequentially a sequence-structure alignment for all pairs of sequences and then combines the pairwise alignments using T-Coffee [Notredame et al., 2000]. The program is still competitive as we have seen in chapter 7, however it is not well maintained in the sense that old libraries are used (e.g. LEDA [Mehlhorn et al., 1996] for access to general matching algorithms) and the code is not parallelized. Hence, my PhD project is a re-implementation of the core algorithms based on the modern C++ library SeqAn [Reinert et al., 2017].

I have implemented the algorithm for pairwise structural alignments in a new C++ program with the name LaRA 2, which reflects that the underlying model is the one of the original LaRA 1, but has been improved with the techniques described in part II of this thesis.

The new tool computes structural alignments with pseudoknots in high quality. It is capable of processing large data sets because of its enormous speed-up thanks to its implementation optimized for multithreading and vectorization. According to the benchmarks LaRA 2 is up to 130× faster than LaRA 1, while maintaining the accuracy. In contrast to existing software it can handle arbitrary pseudoknots and shows better performance on both simulated and experimentally determined RNA structures.

In addition to the multithreading and vectorization techniques, the speed-up of LaRA 2 is due to the new efficient routines for alignment and matching. I have made the SeqAn alignment amenable for position-specific score and added two efficient methods for maximum weighted matching: the blossom algorithm, which computes slightly better matchings, and a faster greedy approach.

For multiple structural alignment I have created two workflows. There is the classic one with T-Coffee, and an improved one with MAFFT X-INS-i, which has proven to be faster and requires less memory.

For versatile input options, I have implemented a wide range of formats for sequences with structure annotation, as well as parsing the dot plot files from RNAfold. Furthermore, together with Gianvito Urgese I have designed the new *Ebpseq* format, which can be used to store sequence and structure information from various sources, including SHAPE data.

Alongside the program I have developed an interactive Jupyter notebook that serves as a template for getting started and provides practical use cases and code for benchmarks. Furthermore, the manual on https://seqan.github.io/lara (22.08.2022) provides assistance for using LaRA 2 with T-Coffee or MAFFT for multiple structural alignments and demonstrates the supported input and output formats.

8.1 Outlook

The performance of LaRA is quite dependent on a good base pair probability prediction. In this section I want to suggest improvements to achieve a better and more realistic structural input for LaRA 2.

A promising addition to the proposed LaRA 2 workflow is the incorporation of additional, experimentally gained structures that have been obtained by SHAPE experiments. This is a type of experiment for RNA that determines the reactivity of the 2'-hydroxyl group in the ribose ring. A high reactivity is found at single-stranded and conformationally flexible positions, while the reactivity is low for base-pair constrained nucleotides [Spitale et al., 2014]. Although the reactivity values are not yet used as additional constraints in LaRA 2, they can already be incorporated via the *Ebpseq* format.

An interesting project to consider is the consensus structure module (Cosmo) from a master thesis by Resta, 2018, which is able to include secondary structures resulting from the three tools IPknot [Sato et al., 2011], RNAfold [Lorenz et al., 2011], and RNAstructure [Reuter and Mathews, 2010]. It combines all the input data in a graph, and aims to create a structural conformation of an RNA family that is supported by multiple tools. It produces an *Ebpseq* file, which LaRA 2 can take as input. However, Cosmo does not use the full capabilities of this format, since it outputs just a single secondary structure into it. In general, a fixed structure performs worse than a base pair probability matrix, because the alignment is forced to satisfy the one given structure, which makes it is extremely sensitive to artefacts or other errors. I suggest improving Cosmo in a way that it passes the unfiltered information to LaRA 2 such that also suboptimal structures can be considered with a given probability.

8.2 Availability

LaRA 2 project

Homepage: https://seqan.github.io/lara
Source code: freely available on https://github.com/seqan/lara
System requirements: tested on Linux and MacOS; gcc or clang compiler
Software dependencies: SeqAn 2.4, Lemon 1.3.1, ViennaRNA 2.0 or higher
License: BSD-License (3-clause)

Benchmark data

Plastids data: http://combio.pl/rrna/download
Reference alignments for pseudoknotted RNA structures: https://rfam.xfam.org
Data and scripts for the BRAliBase 2 benchmark:
http://projects.binf.ku.dk/pgardner/bralibase/bralibase2.html

Part III

MaRs: Motif-based aligned RNA search

9 Structural motifs for classifying ncRNA

Under a structural motif we understand a set of descriptors, which define the properties of an RNA family with respect to sequence and secondary structure. We have seen in section 2.3 that a very flexible way of describing the relevant properties are stem loop descriptors. They divide the molecule into logical units that can be treated independently and naturally support pseudoknots. This chapter describes in detail how stem loops are detected from secondary structure (section 9.3) as well as how the descriptors are designed in `MaRs` (section 9.4).

For creating structural motifs we need two ingredients: a sequence-structure alignment and at least one consensus secondary structure. The alignment can be obtained with `LaRA 2` or other structural aligners, e.g. the ones introduced in section 2.2. Resources for existing alignments are for instance the Rfam database [Kalvari et al., 2018] or the SILVA ribosomal RNA gene database [Quast et al., 2013]. A consensus secondary structure describes the structural conformation of a whole ensemble of related RNA sequences, i.e. the base pairings of the columns of a (multiple) sequence-structure alignment. The consensus secondary structure is either already given inside the alignment file, e.g. using the *Stockholm* format, or otherwise it must be computed from the alignment. Section 9.2 deals with obtaining such a consensus structure.

9.1 Reading and storing a multiple structural alignment

`MaRs` takes as input a structural multiple alignment in either *Clustal*, *Fasta*, or *Stockholm* format. The latter usually contains already a consensus structure (especially if it has been obtained from Rfam) and thus the structure prediction can be skipped. *Clustal* and *Fasta* are the output formats of multiple alignment tools, like `T-Coffee` or `MAFFT`, which we have discussed in chapter 6. They are not able to store a secondary structure, and thus it has to be computed with structure prediction programs. This is similar to the structure prediction in `LaRA 2` (section 2.1), however with the difference that here we want to fold an alignment and not a stand-alone sequence.

A multiple structural alignment in `MaRs` is stored according to the class diagram in figure 9.1. Since the aligned sequences contain gaps, I use the `seqan3::gapped`

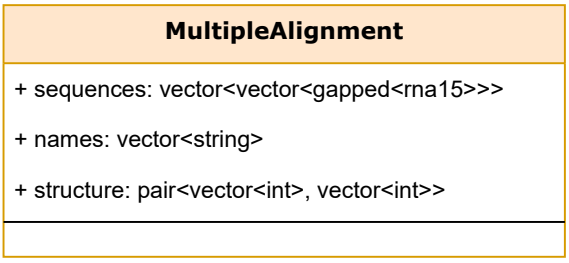


Figure 9.1: **UML representation of the Multiple Alignment class.** Besides the gapped sequences of the alignment, this data structure also holds the sequence names and a consensus secondary structure. It is implemented as a `struct` with public member variables and no member functions.

```
void parse_structure(std::pair<std::vector<int>, std::vector<int>> & structure,
                     std::vector<seqan3::wuss51> const & wuss_string)
{
    structure.first.resize(wuss_string.size(), -1);  // initialize structure with -1 (unpaired)
    structure.second.resize(wuss_string.size(), -1);

    std::stack<int> brackets[seqan3::max_pseudoknot_depth<seqan3::wuss51>];
    int pos = 0;
    for (seqan3::wuss51 symbol: wuss_string)
    {
        int const pkid = seqan3::pseudoknot_id(symbol).value_or(-1);

        if (symbol.is_pair_open())
        {
            brackets[pkid].push(pos);
        }
        else if (symbol.is_pair_close())
        {
            if (brackets[pkid].empty())
                throw seqan3::parse_error{"Invalid WUSS string encountered."};

            structure.first[pos] = brackets[pkid].top(); // set righthand position
            structure.first[brackets[pkid].top()] = pos; // set lefthand position
            // reduce pseudoknot level if the stack on the lower level is unused
            int reduced_pk = pkid;
            while (reduced_pk > 0 && brackets[reduced_pk - 1].empty())
                --reduced_pk;
            structure.second[pos] = structure.second[brackets[pkid].top()] = reduced_pk;
            brackets[pkid].pop();
        }
        // no actions for unpaired
        ++pos;
    }
    for (std::stack<int> const & stack: brackets)
        if (!stack.empty())
            throw seqan3::parse_error{"Invalid WUSS string encountered."};
}
```

Listing 9.1: **Parsing the structure from WUSS notation.** The implementation uses stacks for extracting the positions of corresponding brackets per pseudoknot level.

type with the `rna15` alphabet. This alphabet allows 15 different values and enables `MaRs` to parse and process wildcard symbols from the alignment file. The sequence names are kept for the output later on, and the secondary structure in encoded in two vectors: The first vector contains for each alignment column the position of the interacting column, or -1 if unpaired. The second vector contains for each interaction the pseudoknot page (the *page number* was discussed in section 1.4), or -1 if unpaired.

For *Fasta* input, I simply use the `SeqAn 3` sequence parser, which works also for gapped sequences. The *Clustal* and *Stockholm* alignment formats I have implemented in `MaRs`, because the `SeqAn 3` alignment I/O is designed solely for read alignments and thus not compatible. However, my implementations are written in line with the current I/O design and located in header files without external dependencies, so if desired they can be included into the `SeqAn 3` library without effort later on.

The secondary structure in *Stockholm* files is given in WUSS notation. I wrote a function `parse_structure` that converts this notation into the vector-based structure format described above. The function is shown in listing 9.1 and uses a stack per pseudoknot level (i.e. bracket type) in order to keep track of the positions of the opening brackets. Whenever a closing bracket is encountered, the position of its partner can be obtained from the respective stack. Since the WUSS notation does not always apply the lowest possible pseudoknot level, I reduce the assigned level to the lowest that has an empty stack (see lines 25 to 28 of listing 9.1).

9.2 Obtaining the secondary structure of a multiple alignment

A secondary structure is essential for assigning structural features to the motif. In the case of `MaRs`, it is necessary for the stem loop detection as well as for the distinction between stem and loop regions. Thus, the first step is to obtain such a structure for the given multiple sequence-structure alignment.

With `MaRs` the structure is calculated either with `IPknot` [Sato et al., 2011] or `RNAalifold` [Hofacker et al., 2002]. Because `IPknot` is superior in structure prediction and is able to predict pseudoknots [Sato et al., 2011], I have chosen it as the default method. `RNAalifold` is slightly faster (in the scale of seconds), but since the structure prediction is less than 2% of `MaRs`' run time, this argument can be neglected.

From a technical point of view, `RNAalifold` is implemented in `RNAlib` as part of the `ViennaRNA` package [Lorenz et al., 2011]. Since `IPknot` depends on `RNAlib`, the algorithm of `RNAalifold` is available for free.

`IPknot`, however, has quite a list of software dependencies. If missing, they are automatically downloaded during installation in the cmake step and include `RNAlib`, `Contrafold` [Do et al., 2006], `Nupack` [Zadeh et al., 2011] (depends on `Boost`), `MPFR` [Fousse et al., 2007], and optionally `GLPK`.

```cpp
/*!
 * \brief Compute the secondary structure of a given multiple structural alignment (MSA).
 * \param[in] names The IDs of the MSA.
 * \param[in] seqs  The sequences of the MSA.
 * \return two vectors which hold the base pairs and pseudoknot levels.
 */
std::pair<std::vector<int>, std::vector<int>> run_ipknot(std::list<std::string> const & names,
                                                         std::list<std::string> const & seqs);
```

Listing 9.2: **The interface for calling the IPknot algorithm.** The function takes a multiple structural alignment as second argument, as well as the sequence identifiers. It returns two integer vectors encoding the structural interactions and pseudoknot levels.

```cpp
void compute_structure(Msa & msa)
{
    // Convert names
    std::list<std::string> names{msa.names.size()};
    std::ranges::copy(msa.names, names.begin());

    // Convert sequences
    std::list<std::string> seqs{msa.sequences.size()};
    auto && char_seq = msa.sequences | seqan3::views::to_char;
    for (auto && [src, trg] : seqan3::views::zip(char_seq, seqs))
        std::ranges::copy(src, std::cpp20::back_inserter(trg));

    msa.structure = run_ipknot(names, seqs);
}
```

Listing 9.3: **Usage of the IPknot interface.** The containers for sequence names and sequences have to be converted into lists beforehand. In addition, the sequences have to be transformed from `seqan3::dna15_vector` to `string`.

Although it is implemented in C++, `IPknot` unfortunately does not provide an API that allows a programmer the invocation of the structure prediction algorithm from a C++ function.

Thus, I have created a fork of the `IPknot` repository in the `lib` subdirectory of `MaRs`, in which I have removed the `main()` function from `src/ipknot.cpp` and replaced it with a function `run_ipknot()`. It contains the relevant code path from `main()` that invokes the desired algorithm. Essentially, I have removed the code related to argument parsing and file in- and output.

The interface of this function as declared in `src/structure.hpp` is shown in listing 9.2, however I refrain from citing the definition, since it is not my own code. Instead, the function definition can be looked up in file `lib/ipknot/ipknot.cpp` in the `MaRs` repository.

The invocation of the presented `IPknot` interface is shown in listing 9.3. In order to adapt to the interface, I have to change the container type of the sequence names and sequences into linked lists, and transform the sequences into `char` type.

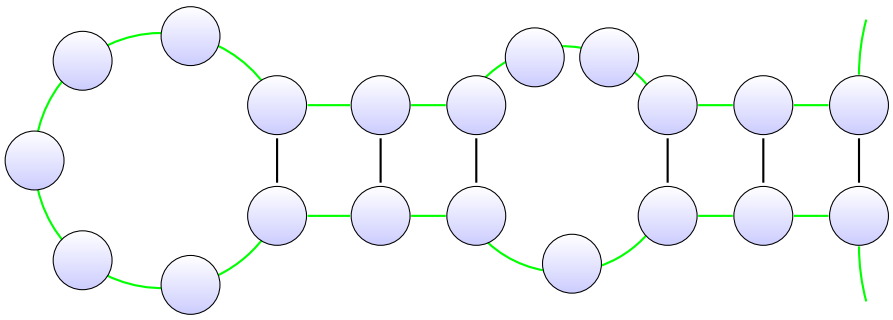


Figure 9.2: **Stem loop structure.** A stem loop consists of exactly one hairpin (on the left) with an associated stem region, which in this case contains an internal loop. The green line marks the RNA backbone, while the black connections denote base-pair interactions. On the right end there could be e.g. a multiloop or two single-stranded ends.

9.3 Stem loop detection

A stem loop can be detected unmistakably through its hairpin loop, since each stem loop has exactly one. Figure 9.2 shows an example of a stem loop, and its hairpin is located on the left-hand side. The hairpin loop is closed with a stem region, which includes at least one stem (consecutive base-pairs), and possibly internal or bulge loops alternating with further stems. Remember that we have discussed the different RNA structure components in section 1.3 on page 8.

The algorithm for stem loop detection scans linearly the RNA backbone from 5' to 3' end, i.e. it iterates the structure vectors introduced in section 9.1. If the structure contains pseudoknots, we have to scan for each pseudoknot level independently, considering only the base pairs that are active for the particular level. Listing 9.4 shows the algorithm for detecting the stem loops within a given level. As an example, for the structure in figure 1.4b we do one run considering only the parentheses `()`, which finds the stem loop of black interactions, and a second run for the squared brackets `[]`, which finds the stem loop of the red ones.

Let bp be the first structure vector, which contains the base pairings. While we iterate at position i in the vector, the value $bp[i]$ tells us, if we are at an unpaired position ($bp[i] = -1$), or at an interaction site with a partner to the right ($bp[i] > i$) or to the left ($\ell \leq bp[i] < i$), with $\ell = 0$ initially.

As soon as we spot the first interaction with $\ell \leq bp[i] < i$, we have found the closing base pair of a hairpin loop. We iterate further, as long as $\ell \leq bp[i] < i$ or $bp[i] = -1$ holds. This process extends the current stem loop region, until we either reach the end of the bp vector, or we encounter an interaction that belongs already to the following stem loop ($bp[i] > i$) or that closes a multiloop ($-1 < bp[i] < \ell$). We report the coordinates $(bp[i], i)$ of the outermost valid interaction of each stem loop and set $\ell = i + 1$ for continuing the search.

The meaning of the ℓ variable, which corresponds to `left` in listing 9.4, is to keep track of the leftmost position for a stem loop to start. This is necessary, because

```
1  void detect_stemloops(Motif & stemloops,
2                        std::vector<int> const & bpseq,
3                        std::vector<int> const & plevel,
4                        int level)
5  {
6      std::pair<int, int> coordinates{-1, -1}; // store bounds of current stem loop
7      unsigned char id_cnt{0u}; // generates unique ids for each stem loop
8      int left = 0; // leftmost start of current stem loop
9
10     // 0-based indices
11     for (auto && [idx, bp, pk] : seqan3::views::zip(std::ranges::views::iota(0), bpseq, plevel))
12     {
13         if (bp == -1  pk != level) // unpaired site or non-matching pseudoknot level
14             continue;
15
16         if (bp < idx && left <= bp) // extend current stem loop
17         {
18             coordinates = {bp, idx};
19         }
20         else if (idx < bp && coordinates.second != -1) // new stem loop after previous is finished
21         {
22             left = coordinates.second + 1;
23             stemloops.emplace_back(id_cnt++, coordinates);
24             coordinates = {-1, -1};
25         }
26     }
27     stemloops.emplace_back(id_cnt, coordinates); // store the last stem loop
28 }
```

Listing 9.4: **Algorithm that detects stem loops** for a particular pseudoknot level. The main loop considers three values per iteration: `idx` is the current position i, i.e. a counter that starts from 0, `bp` is the current value of the base pair vector and corresponds to $bp[i]$ in the text, and `pk` is the pseudoknot level.

interactions that close a multiloop nest more than one hairpin loop, even though they satisfy the $bp[i] < i$ condition.

9.4 Motif design

The type of the first parameter of function `detect_stemloops` in listing 9.4 is a reference to `Motif`, which is simply an alias for a vector of stem loops. We see in line 23 that new stem loops are appended to this vector by in-place construction from a unique identifier number and the boundary position. Thus, the stem loops already contain their positional offset within the alignment, but no sequence-structure information yet. In this section I discuss the retrieval of this information, as well as the design of the `Stemloop` class and its two dependent data structures `LoopElement` and `StemElement`, as visualized in figure 9.3.

After the constructor sets the unique identifier and the boundaries during the stem loop detection step, the remaining member variables of class `Stemloop` are set afterwards by the `analyze` method. It takes the multiple sequence-structure alignment as a parameter (the data structure introduced in sections 9.1 to 9.2)

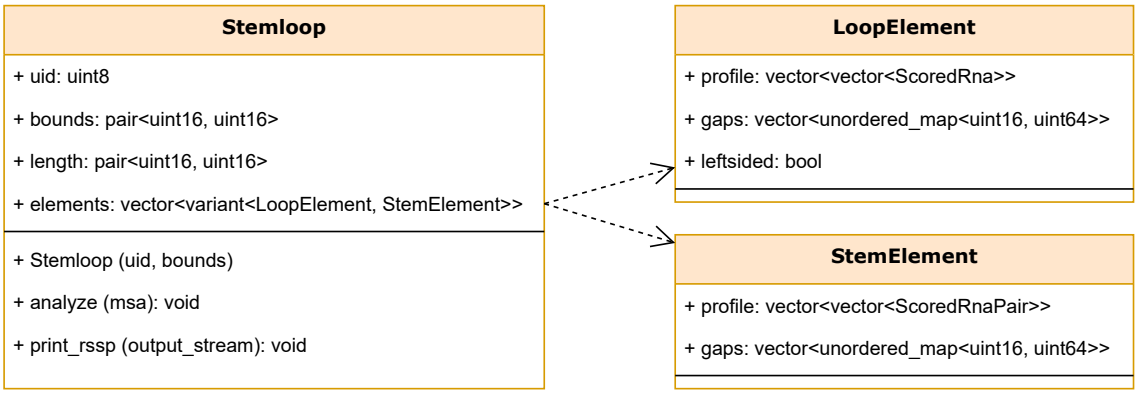


Figure 9.3: **UML representation of the stem loop class** and its dependent data structures `LoopElement` and `StemElement`. Since many members are unsigned integer types of various size, I specify them here with their exact size information, e.g. `uint8` is an 8-bit unsigned integer with a value range between 0 and $2^8 - 1 = 255$.

and uses it to extract the desired information. Since the location of the stem loop within the alignment is already known from the `bounds` member, it operates only on the alignment columns that are located within the respective region. In `MaRs` the stem loop analysis is executed in parallel, because this step can be performed independently for each stem loop.

The `length` member is a pair of two numbers, which denote the minimum and maximum possible length of the stem loop. For each sequence we count the number of nucleotides within the stem loop region and set the minimum and maximum length accordingly. Thus, the minimum length is determined by the stem loop sequence with the most gap characters, while the maximum length is determined by the stem loop sequence with the least gap characters. The length information is used later on to prune the search tree, see items 2 and 4b on page 89.

The `elements` member is a vector that consists of (usually alternating) loop and stem elements. A loop element represents a single-stranded region from a hairpin, bulge, or internal loop, while a stem element stores information about stems. My implementation uses the STL class template `std::variant` for storing the alternating types in a single vector.

The first element of the vector is the hairpin loop, which we know is always present and is supposed to be searched at first. The only occurrences of two adjacent loop elements in the vector are caused by internal loops, which are stored as two separate loops, the left-sided (5') and the right-sided (3') one. The information about the orientation of each loop is stored in the `leftsided` flag of the loop elements, see figure 9.3. Note that I have defined each hairpin loop right-sided.

In the `profile` of a loop or stem element is encoded which nucleotides occur at each alignment column and with which likelihood. The outer vector has the length of the alignment and each element contains information of one alignment

column. The inner vector contains the set of the possible nucleotides with associated likelihood, sorted in descending order by likelihood. I have also tried to use a `std::set` here, but it has turned out to be more efficient to sort the data once it has been collected in a vector, since it has faster access times, which is desired for the search. For a loop, the internal type is `ScoredRna`, which resolves to a pair of a float value and a single nucleotide of type `seqan3::rna4`. In case of a stem, the internal type is `ScoredRnaPair`, where the second type of the pair is a pair of two `seqan3::gapped<seqan3::rna4>` characters.

Gaps are in general stored apart from the sequence profiles, various good reasons have been reported by Strauch, 2017: The individual storage of single gap symbols would lead to a combinatorial problem in enumerating the patterns, and, even more important, lose information about the length of a gap streak, i.e. one deletion of length 5 in a sequence is different from five deletions of length 1 scattered across different sequences and columns.

The `gaps` variable is a vector that contains for each alignment column c a hash map of gaps that start in column c. The hash key k is the length of the gap streak, and the assigned value is the number of gaps of length k that start in c.

However, for stems I differentiate between two types of gaps: a two-sided gap has a gap symbol on both sides of the stem and is stored in the `gaps` data structure; a one-sided gap is a gap symbol that has a structural interaction with a nucleotide and is stored in the profile. The reason is that one-sided gaps are very rare and usually have a length of just 1, so it does not lead to combinatorial problems in this case, and furthermore keeping the nucleotide information allows for a better representation of the RNA family.

Before I go into details about the profile creation in the following subsections, I want to mention the function `print_rssp`, which is also shown as a member of the `Stemloop` class in figure 9.3. This function generates output for a pattern file (`.pat`) that contains the stem loop structure in dot-bracket notation alongside a single sequence string. The sequence string contains the wildcard character `N` where the profile contains multiple options, and a nucleotide if the profile is unique at a site. The file that results from concatenating all the stem loops' patterns is used for benchmarking the `Structator` tool, see section 12.1.

9.4.1 Obtaining a loop profile

For counting the occurrences of the different characters in an alignment column, I implemented a template class `profile_char` that works for all alphabets that model `seqan3::semialphabet`. In particular, for loops I instantiate it with the `seqan3::rna4` alphabet, so internally it stores an array of size 4 with the counts for `A`, `C`, `G`, and `U`. It has a function `increment`, which takes a character from the alignment and adds to the respective count(s), unless it is a gap. If one of the four nucleotides is given, one of the counts in increased by 1. However, if a wildcard character is given that substitutes for k different nucleotides, the respective k counters are increased by $\frac{1}{k}$ each.

A	0	40	0	0	50	25
C	100	0	60	70	10	25
G	0	10	40	30	20	25
U	0	40	0	0	20	25

A	-14.14	0.42	-14.14	-14.14	0.73	-0.26
C	2.32	-13.55	1.59	1.81	-1.00	0.32
G	-13.55	-1.00	1.00	0.59	0.00	0.32
U	-14.14	0.42	-14.14	-14.14	-0.58	-0.26

Table 9.1: **Example of a loop profile.** Each column in the table corresponds to a column of an alignment with $n = 100$ sequences. The top half of the table shows the absolute nucleotide counts c_z as collected in the profile chars, the bottom half the corresponding logarithmic odds scores s_z, assuming the expected nucleotide distribution $e_A = e_U = 0.3$ and $e_C = e_G = 0.2$. The second column must contain 10 gap symbols, since $\sum_z c_z = 90$.

The sequence profile for a loop is created by scanning the respective alignment columns. When a column is processed, an empty profile char is created and for each of the column's symbols the `increment` function is called. Afterwards, the collected quantities can be queried from the profile char and transformed into score values, as shown in table 9.1.

The transformation into score values is important, since we want an additive score that is not dependent on the profile length, and we want to model the statistical significance of the nucleotides present in the alignment in relation to their expected distribution. For satisfying the second property, I compute an *odds score*, which is the quotient of actual and expected occurrence. The additive property can be met by transforming the score into the logarithmic space. Thus, the score s_z of nucleotide z with expected relative occurrence e_z is computed as follows:

$$s_z = \log_2 \frac{r_z}{e_z} \quad \text{with } r_z = \frac{c_z + \varepsilon}{n}$$

The term r_z is the relative occurrence, i.e. the absolute count value c_z from the profile char divided by the number of sequences n. In the shown formula, I have added a pseudo count of $\varepsilon = \frac{1}{600}$ to the c_z value to avoid that the argument of the logarithm is zero. I have taken the e_z values from Olson et al., 2009 and since these values are constant, `MaRs` pre-computes their logarithms:

$$s_z = \log_2 \frac{c_z + \varepsilon}{n} - \log_2 e_z$$

We can see from the formula for the logarithmic odds score that counts above the expectancy result in a positive score and vice versa: $r_z > e_z \Leftrightarrow s_z > 0$. This means that a few uncommon occurrences can easily generate high scores and thus gain a lot of attention for the search step. On the one hand this is desired to model

the statistical significance, on the other it can easily lead to mistakes caused by misalignments or sequencing errors.

The gaps profile is created together with the sequence profile while scanning the alignment columns. A vector *gp* is used for storing the start positions of gap streaks for each sequence. The initial value for each entry is -1, which means that currently no gap is present. If a gap symbol is encountered in sequence i and $gp[i] = -1$, then we have spotted a gap opening and the respective column number is recorded in *gp*. As soon as the column occurs during the scan where there is no gap symbol in sequence i any more, the gap is recorded with its length (the difference of the column indices) and start position (the value of $gp[i]$) in the hash map vector described above. If the hash map key already exists, we have found another occurrence of the same gap in a different sequence, and thus the counter value of the existing map entry is incremented. Since the gap has been closed, the value of $gp[i]$ is reset to -1.

9.4.2 Obtaining a stem profile

In a stem we want to observe the occurrence of nucleotide pairs as well as one-sided gaps. I have implemented in `MaRs` an alphabet type called `bi_alphabet`, which for any alphabet α represents the combined $\alpha \times \alpha$ alphabet. Since I want to represent one-sided gaps as well (see above), the particular type for the stem alphabet is `bi_alphabet<seqan3::gapped<seqan3::rna4>>`. It has an alphabet size of 25 and models all the possible pairs that can occur in a stem: `AA`, `AC`, `AG`, `AU`, `A-`, `CA`, ..., `-G`, `-U`, and `--`. Note that the last value exists for technical reasons and is never used, since two-sided gaps are stored in the gaps data structure instead.

For obtaining the stem profile, I use the `profile_char` class that I have introduced already for the loop region. Since the described stem alphabet models `seqan3::semialphabet` as well, it is possible to apply the same methods as above. The internal array has 25 counters in this case, and it automatically deals with wildcards in the `increment` function. The score calculation uses the same formulae as for loops, but with z denoting a pair of nucleotides instead. The e_z values of the expected distribution of base pairs I have taken from Olson et al., 2009, however the values for gap pairings are not available. I have set them to 1, because small values would add more significance to gaps than desired, while the value 1 is rather neutral with $\log_2 1 = 0$.

9.4.3 Pruning the sequence and gap profiles

In order to avoid too many search paths, I have initially tried to implement a filter based on the relative occurrence r_z, which failed because the filter either did not remove anything, or the search sensitivity dropped significantly. After I had changed to scoring system towards the logarithmic odds score as described in the previous subsections, I came up with a new filter, based on the odds score, i.e. the quotient of actual and expected frequency.

The filter has a parameter p, which defaults to 10. From the sequence profiles (of loops and stems) the filter removes the nucleotides z, which occur less than $p\,\%$ of the expected frequency, i.e. if $r_z < e_z \cdot p\,\%$.

Given that $r_z \geq \frac{1}{n}$ and $e_z \leq 0.5721$ (the expected frequency of the most likely nucleotide pair, which is `GC`) we follow from $\frac{1}{n} < 0.5721 \cdot p\,\%$ that $n > \frac{100}{0.5721 \cdot p}$, so the sequence filter is effective with at least $\frac{175}{p}$ sequences in the alignment.

An exception are zero-counts, which are not yet considered, since $r_z = \frac{\varepsilon}{n}$. The inequation $\frac{\varepsilon}{n} < e_z \cdot p\,\%$ is in practice almost always true for $p > 0$, so zero-counts are removed from the motif. However, for $p = 0$ the inequation is false and the motif keeps its full content.

For gaps, since we do not have reliable expectancy values, the filter removes a gap if it occurs in less than $\frac{p}{2}\,\%$ of the sequences. Thus, the gaps filter is only effective for alignments with more than $\frac{200}{p}$ sequences.

We can see that apart from removing zero-counts this implemented filter method prunes only in alignments with many sequences. This is very appreciated, since small alignments are fast anyway, and we want the filter to operate on large alignments to compute them in a feasible time. As we see in the benchmarks (chapter 12), disabling the filter with $p = 0$ leads to a large run time that is spent mainly with a few alignments that can be characterized either by their enormous number of sequences or by the presence of a very long stem loop. Furthermore, a disabled filter keeps even zero-counts, which creates an exponential number of considered search paths (4^L for loops and 24^L for stems of length L). If a high sensitivity is important, I recommend setting $p = 2$.

10 Motif search in an indexed genome

In this chapter I describe the methods that `MaRs` uses to locate an RNA family in a genome. I put genome, because it is the intended application for this method, however it can be applied to any kind of sequence. Furthermore, `MaRs` is not limited to a single (genomic) sequence, instead it works as well for a set of sequences, e.g. taken from a sequence database.

A detailed description of an RNA family's properties is provided with the motif, which we have discussed in the previous chapter. The motif consists of a set of stem loops, which are subject to independent searches. In the first section of this chapter I describe the data structure that allows us to perform fast searches, and section 10.2 discusses how a stem loop search is conducted with the help of this data structure. The searches generate so-called hits, which are genome positions where a stem loop matches well. In the final step, which is covered in section 10.3, a linear scan over the genome reveals the locations, where hits from different stem loops can be grouped into matches for the whole motif.

10.1 Bi-directional index

Searching multiple queries in a genome in a naïve way is not a good idea, since the run time is proportional to the genome length, the number of queries, and the average query length. As an alternative, an additional data structure called index can be used to store e.g. pre-computed locations of substrings in alphabetical order. This reduces significantly the computational cost of each search, since the run time is not any more proportional to the genome length, but to its binary logarithm.

The cost of using an index is its additional memory usage and the time for its construction. For keeping the memory footprint as low as possible, I use a bi-directional FM-index in `MaRs`, as discussed already in section 2.3. The benchmark in section 12.1 demonstrates that the FM-index requires significantly less memory and a shorter construction time compared to the affix tree used in `Structator` [Meyer et al., 2011]. The construction step of the index is required only once, as long as the sequence does not change. The index can be stored alongside the sequence in a separate file, so it is persistent across different program runs, and it is even transferable among different computers.

Like the `Structator` program, `MaRs` applies a bi-directional index, which is able to expand the search pattern not only to the right, but in both directions. This

is a great benefit for structural RNA search, since a hit can be expanded in both directions in order to check for the complementary base pairs in the stem regions.

The relevant data structure for searching a query string in the bi-directional FM-index of SeqAn 3 is the *cursor*. It represents the query, which is initially empty, and can be extended gradually with nucleotide characters to the left and right. Instead of storing the query as a string, the cursor maintains pointers that mark the interval in the internal table of the index where the query is represented. The size of this interval corresponds to the number of occurrences of the query in the genome. Thus, the interval becomes successively smaller whenever more nucleotides are appended (`extend_right`) or prepended (`extend_left`) to the query. The extend-operations of the SeqAn 3 cursor return whether the extension was successful, i.e. the query could be extended such that it exists at least once as a substring in the indexed genome.

As input parameter for the genome MaRs accepts the name of a sequence file. My implementation uses `seqan3::sequence_file_input` for parsing the file, which supports the following formats: *Fasta*, *Fastq*, *Embl*, and *Genbank*. But before the file is actually read, the program checks whether an index file exists in the directory of the specified file.

If an index file is present, the index is loaded from that file and the sequence file is ignored (it does not even need to exist). The index file name is the sequence file name with the additional suffix `.marsindex`.

In case an index file is absent, MaRs parses the given sequence file and computes an index. With SeqAn 3 this as easy as calling the constructor of `seqan3::bi_fm_index` and passing the sequences as argument. In the end, MaRs writes an index file next to the location of the sequence file such that the index is available much faster in the future.

I have implemented the index in- and output with the `Cereal` library and use a binary encoding. Binary files are not readable with text editors, but they have much smaller file sizes. Alongside the index the file stores a version string (in case the format needs to be changed in the future) and the sequence name(s). The sequence names are usually short compared to the sequence content and add the benefit that the index file is self-sufficient and thus MaRs does not need to parse the sequence file only for the sequence names, which are needed for the output of the results.

In addition, I have added a flag `-z` to MaRs that controls whether the index output shall be compressed. Compression creates an even smaller file with the drawback that the decompression takes additional time whenever the file is read from disk. The compressed index file has the suffix `.marsindex.gz`. I have implemented the support for compressed index in- and output with the *gzip* support streams of SeqAn 3. If, for whatever reason, an uncompressed and a compressed index are both present for reading, the uncompressed one is preferred since it loads faster, as the benchmark in section 12.1 demonstrates. Note that users may also use the `gzip` program to (de)compress the index files outside MaRs.

The index data structure is created for the `seqan3::dna4` alphabet and uses the `collection` layout, since I want MaRs to support input files that possibly contain multiple input sequences. Internally, the index concatenates all the sequences with

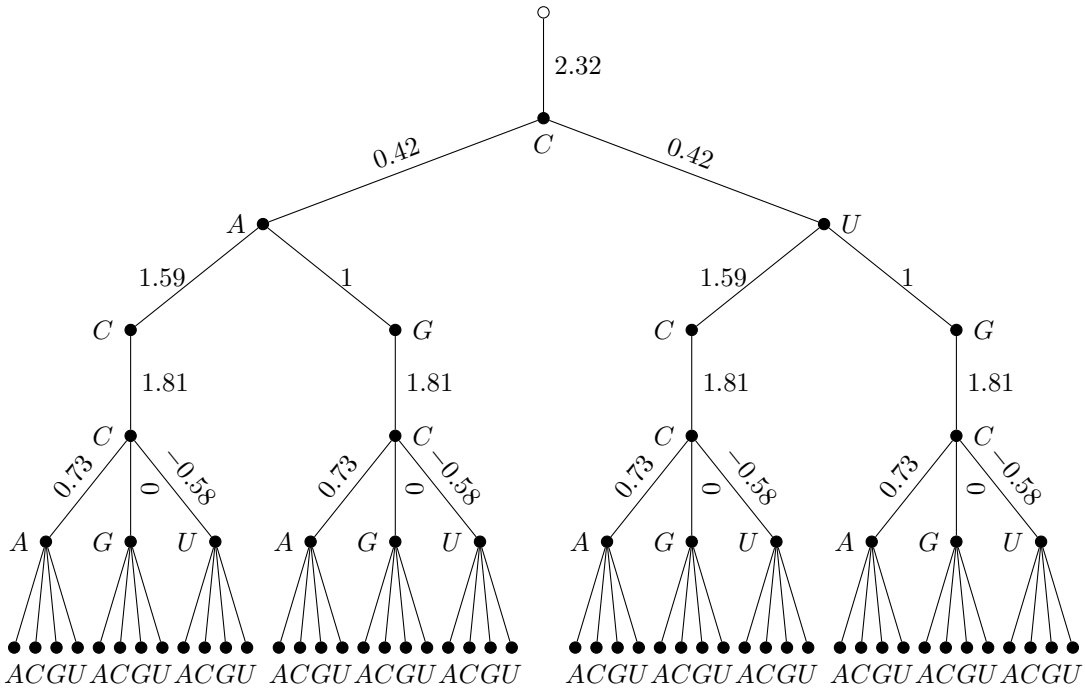


Figure 10.1: **Motif search scheme.** The tree shows the different search paths with scores for the loop region from table 9.1. Some low-scoring branches are omitted for the sake of a clearer arrangement.

sentinel characters in between and keeps track of their positions. Thus, the search results that we see in the following sections consist of the sequence number alongside the position.

10.2 Finding stem loop hits

In this section I describe how we use the index to find the positions where the stem loop descriptors match (so-called *hits*). As discussed above, we start to generate a search query at the hairpin loop and extend it in both directions until we reach the outermost base pair of the stem loop. On the way from inside to outside we need to consider all the alternative options of the profile, and since there may be multiple branches at each site, the search procedure actually looks like a tree.

Figure 10.1 visualizes such a search tree for the loop profile example that we have already examined in table 9.1 on page 81. The first column of the profile contains the `C` nucleotide with score 2.32, and some zero-counts that we consider to be filtered out. Thus, we initiate a search for this character and obtain a cursor to the index, which includes all the positions in the sequence that contain the character `C`. In a vector called `history`, which is still empty at this point, we store the current score (2.32) and the cursor.

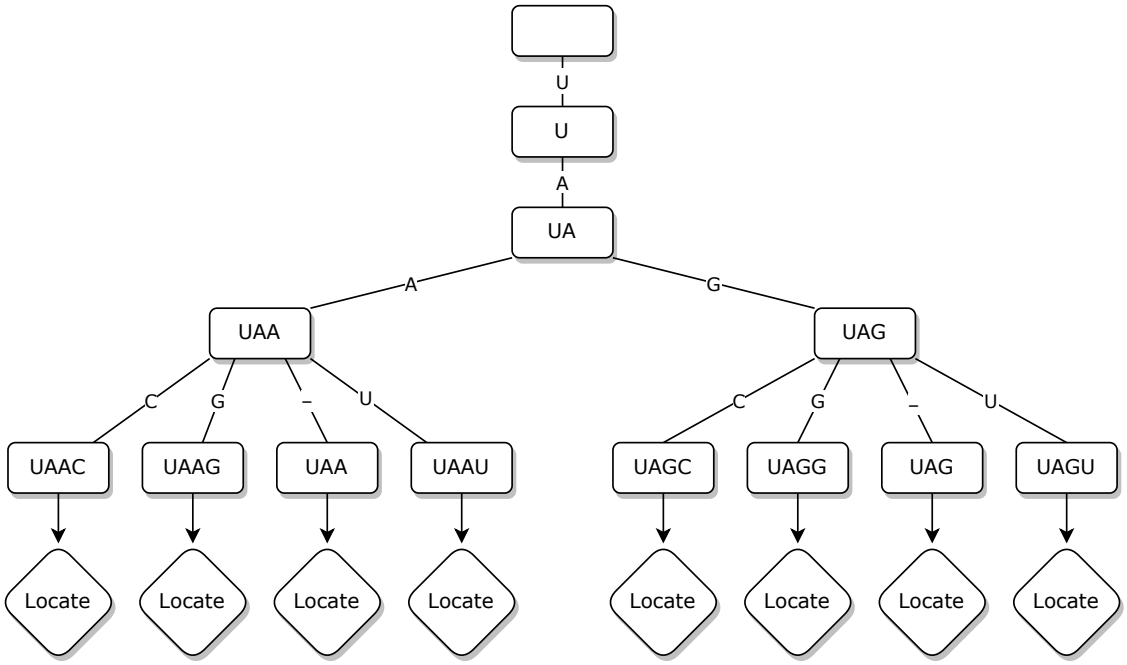


Figure 10.2: **Index search with backtracking and gaps.** The nodes of this search tree demonstrates the evolution of the search query. Whenever the algorithm reaches a leaf node, the locations of the query in the genome are generated.

We visit the next column of the profile and in this description let us consider the two options `A` and `U`. The search follows the depth-first paradigm, so we visit the whole sub-tree of `A` first, revert to this state with the help of the history vector, and visit the remaining sub-trees (according to the score in descending order). Since the actions for each node are the same, I implemented them in a recursive function.

Returning to the example, the recursive function creates a copy of the last cursor and appends the `A` to the right (assuming a right-sided loop, cf. section 9.4), so the new cursor includes all the `CA` substrings in the genome. The new cumulative score is $2.32 + 0.42 = 2.74$, which is the previous score from history plus the current one. Finally, the new cursor and cumulative score are appended to the history vector.

Whenever we have arrived at a leaf node, i.e. the end of the stem loop, a function `locate` is called, as depicted in figure 10.2. Based on the current cursor, it extracts all the genome locations from the index where the current search string matches. Each location alongside the cumulative score, the length of the search string, and the stem loop identifier build a *hit*. The generated hits are stored in a data structure that I am going to discuss in the next section.

We need to backtrack to the previous level if we are either at a leaf node or if all the sub-trees of the current node have been visited. Backtracking is very easy in my implementation, since we only need to erase the last entry of the history vector to restore the previous state.

The procedure for stem regions is the same as for loop regions with a single exception: we append two characters to the cursor per node. Since each stem element provides a character pair, we append the first one on the left side, and the second one on the right side of the search query. This stem-based expansion is the crucial step that uses the conserved structural interactions for reducing the search space effectively.

Alongside the sequence profile we have to check for gaps once at each node of the search tree. Therefore, we iterate the respective hash map of the `gaps` data structure (cf. figure 9.3) and execute the following for each gap steak length L: We continue the search after leaving L columns of the profile out. This can be depicted as appending additional sub-trees to the current node, which consist of the remaining profile after skipping L entries. In the implementation this causes additional calls of the recursive function, where the iterators of the `profile` and `gaps` vectors are advanced by L, while the cursor and score remain the same.

Finally, let us discuss the various conditions of pruning the search tree. This means that if one of the following conditions occur, the search is aborted, we return to the previous state in history (backtracking), and continue the search from there.

1. The cursor returns false on the append operation, i.e. the query sequence does not exist in the genome.

2. The maximum length of the stem loop has been exceeded. This scenario arises if the alignment contains gaps in all the sequences, but the search path has not considered any or very few gaps yet.

3. The score is lower than d search steps ago. The assumption is that a series of rather unlikely choices from the profile does not lead to a good search string any more. Thus, the algorithm checks whether the current score is less than d entries back in the history vector. The default value in `MaRs` is $d = 4$, and can be changed with a user parameter.

4. At the end of a stem loop (i.e. at a leaf node) the `locate` step is omitted if
   a) the cumulative score is negative,
   b) the query does not reach the minimum length of the stem loop because too many gaps were included, or
   c) the query length is shorter than 6, since we would generate too many spurious hits and furthermore such a short stem loop would be sterically very unlikely.

In general, these conditions are heuristics that aim to reduce the run time of `MaRs` by avoiding search paths that likely do not lead to good solutions. After I had found out that the `locate` function is the most time-consuming step of the search, the overall run time has improved very much with the implementation of the conditions in item 4. This is also respected in the parallel implementation of `MaRs`, which I discuss in chapter 11.

10.3 Generation of motif matches

We have seen in the previous section that the `locate` function generates so-called *hits*. A hit defines the occurrence of a particular stem loop in the genome and contains the genome location, cumulative score, query length, and stem loop identifier. Since the genome may exist in the form of multiple sequences, the location consists of a sequence identifier, which is the positional index i for the ith sequence, and the start position within this sequence.

Hits are collected in n different vectors, where n is the number of sequences. The assignment to the vectors is based on the hits' sequence identifiers i, so after the search we have assembled all the hits per sequence. This has three advantages:

1. We do not need to store the sequence identifier in each hit.
2. The pooling step is much simpler if nearby hits are stored together.
3. The hit vectors are independent, thus pooling can be multithreaded well.

Since I describe the multithreading in the next section, let us focus on the pooling step here. The aim is to group nearby hits of different stem loops into *matches*. A match is a location in the genome that fits very well to the whole motif and thus is considered an occurrence of the RNA family.

The first step is sorting the hit vector by the compensated positions of the hits. A compensated position is the detected start position of the hit in the genome minus the offset of the stem loop within the alignment, which we have stored in the first bounds coordinate of the stem loop (see figure 9.3). This compensation has the effect that the hits are mapped to the same start position of the motif, no matter from which stem loop they originate.

In the second step we want to detect clusters and therefore iterate the hit vector in order. We start with the first hit and add further hits to the cluster, as long as their compensated position is less than $\frac{1}{2}L$ away, where L is the alignment length. As an additional condition, in a cluster we collect only one hit per stem loop, which is (if multiple hits occur) the hit with the highest score. If a hit is encountered that exceeds the allowed range for the cluster, a new cluster is initialized with this hit.

In every cluster we collect the following data for generating a match:

diversity the number of different stem loops

start position the smallest hit position (not compensated)

end position the largest value for the sum of hit position + query length

score the sum of the hits' scores

sequence identifier known from the hit vector we are processing

query length the sum of the hits' query lengths

$$\textbf{e-value} = \frac{\text{genome length} \cdot \text{query length}}{2^{\text{score}}} \quad \text{[Altschul et al., 1997]}$$

The last step filters and prints the collected matches. The output is a column-based format, inspired by the output formats of `Structator` and `Infernal`. This is to ease the integration of `MaRs` into existing workflows. The columns contain from left to right: sequence name, numeric sequence identifier, start position, end position, query length, diversity, score, and e-value. For filtering low-scoring matches, I have implemented in `MaRs` two methods that can be switched with a parameter.

One method is based on e-values and is the default method in `MaRs`. An e-value represents the significance of a match, i.e. the likelihood that the genome and the query produce a match of the given score by chance [Altschul et al., 1997]. Thus, small e-values are preferred, and matches with e-values smaller than 10^{-10} always pass the filter. But occasionally, there are no such good e-values in the set and an empty output is also not desired. Let E_{min} be the lowest e-value in the set of matches; the matches with an e-value $< 10\sqrt{E_{\text{min}}}$ additionally pass the filter. Such possible ranges are for instance $10^{-10} \dots 10^{-4}$ or $10^{-6} \dots 10^{-2}$.

Another method is rather experimental, since it has not shown as good results yet. This filter consists of two conditions that both must be satisfied for a hit in order to pass the filter: diversity $> \left\lfloor \frac{1}{4}k \right\rfloor$ with k the number of stem loops, i.e. more than one quarter of the stem loops must be found, and score $> k \cdot s$, where s is a user parameter for the average score that stem loop should achieve. I found that the parameter s is hard to set without prior knowledge about the data, for the benchmark in section 12.5 on Rfam data I was using $s = 0.25$.

11 Multithreading in MaRs

In general, MaRs has many individual tasks that can be executed in parallel, a good overview is given in figure 11.1. The amount of tasks and their size depend on various circumstances, which are not known a-priori, like the number of stem loops in the predicted secondary structure, the availability of an index, the amount of leaf nodes in the search tree, and the number of genome sequences.

11.1 Thread pool

For decoupling the tasks from the system specifications, especially the number of available compute cores, I wanted to use a *thread pool* that automatically assigns the tasks to threads and computes them in parallel. The idea is to submit one or more tasks to the pool, which are kept in a waiting queue until computational resources are free to execute them. On submission of a task, the main thread receives a so-called *future*, which is able to wait for the task to finish and to access its return value. In MaRs the number of available worker threads can be set with the `--threads` parameter (which by default uses all available threads on the system).

The choice of a suitable and efficient thread pool implementation was not very easy, and I have tried various options. My first idea was to check in SeqAn 3, and I found the `execution_handler_parallel`. Unfortunately, it does not provide the flexibility of submitting different tasks and e.g. waiting for a particular task — I conclude that its design is dedicated mainly to the alignment and search algorithms within SeqAn. The `ThreadPool` implemented by Nathaniel J. McClatchey[1] failed to run subtasks, i.e. a task created by another task, and otherwise it was rather slow. Andrey Kubarkov provides the `thread-pool-cpp`[2], which has a rather minimalistic API and is very fast, since the tasks are detached, but the drawback is that it lacks a mechanism to query whether a task in the pool has been finished. A promising implementation is `riften::Thiefpool` by Conor Williams[3] that uses a lock-free queue and very modern C++ code. What prevents me from using this one is that it requires the C++ 20 standard and adds more dependent software libraries. Since MaRs is implemented in C++ 17 and still supports gcc versions ≥ 7, I decided that I do not want to give up on this compatibility.

[1] https://github.com/nmcclatchey/ThreadPool (06.04.2022)
[2] https://github.com/inkooboo/thread-pool-cpp (06.04.2022)
[3] https://github.com/ConorWilliams/Threadpool (06.04.2022)

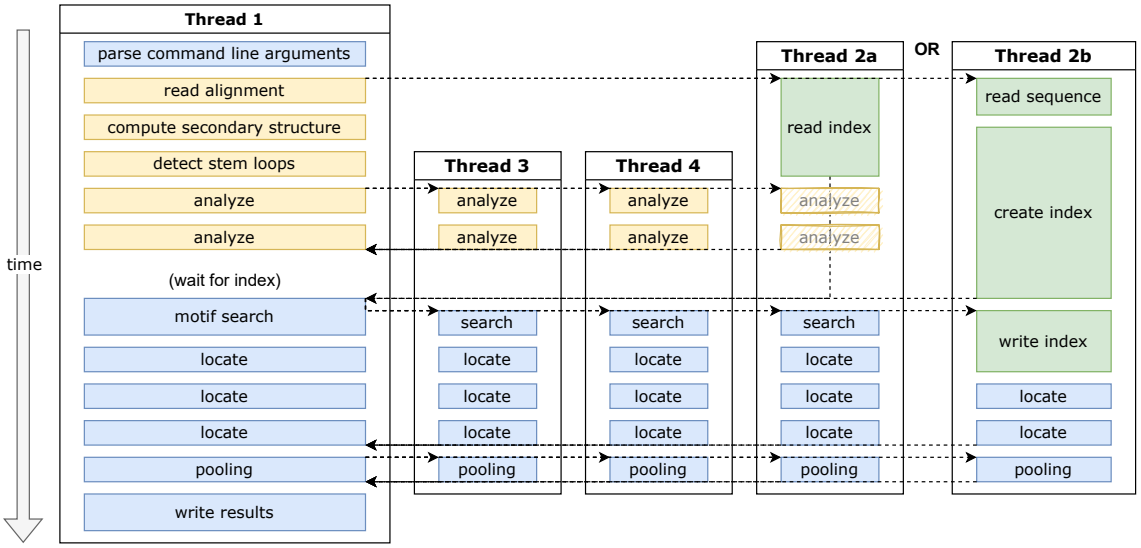


Figure 11.1: **Multithreading scheme for MaRs.** The figure shows a possible program flow for 4 threads. More threads can be imagined as copies of threads 3 and 4, so the work of the stem loop analysis, the hit location, and the pooling would be distributed over more threads. The yellow tiles correspond to the motif preparation, the green ones to the index creation. Thread 2 is depicted in two versions to demonstrate the difference of an available or absent index file.

Finally, I decided to use the `C++ Thread Pool` by Dmitry Danilov[4], which is a lightweight single-header implementation of a thread pool, which is still efficient and easy to use. After construction with the desired thread count I keep an instance of the thread pool during the whole run time of `MaRs`. The central function for adding a task to the queue is the `submit` function, which accepts any type of function and returns a future object.

11.2 Motif and index construction

For performing the search, `MaRs` needs the structural motif in the form of fully analysed stem loops, as well as the constructed index. These two ingredients can be created completely independently of each other, as figure 11.1 shows in yellow and green tiles.

The preparation of the stem loops includes reading the alignment, computing secondary structures, detecting the stem loop boundaries, and analysing the stem loops. The latter can be shared among all available threads, since the stem loops are already constructed and just need to be completed based on their assigned slice of alignment. Therefore, the analysis tasks are submitted to the thread pool.

[4] `https://github.com/f-squirrel/thread_pool` (06.04.2022)

Simultaneously, the index is constructed in its own thread. The corresponding task is submitted to the thread pool immediately after the command line parsing, so it definitely starts before the stem loop analysis. The required amount of work depends heavily on whether the index just needs to be read from a file (thread 2a), which is rather fast and depending on the genome size it finishes before or after the stem loop analysis. In the other case, the index must be computed after reading a sequence file (thread 2b), which always takes much time, so the main thread usually has to wait for the index to be ready.

Since the search algorithm needs the stem loops as well as the index, a synchronization point is required to wait for the slower part. This is implemented with calling `wait()` on the future objects that have been returned on task submission.

11.3 Parallel motif search and match generation

We have seen in section 10.2 that each stem loop spans its individual search tree during the recursive generation of search queries. Thus, my first version of a parallel search implementation was submitting a search task for each stem loop to the thread pool. This worked but did not gain much speed-up. Analyses of the thread usage revealed that after about 10 seconds only one or sometimes two threads were actually working, while the majority of the search tasks have already finished. In the extreme cases, `MaRs` spent 2 hours or more on searching one stem loop with a single thread.

My attempt to solve this problem was the design of tasks that solve individual branches of the search tree. This attempt failed, since the data management between the recursive tasks became quite complex and there was almost no improvement of runtime. For a better balancing of workload I have experimented with task-stealing thread pools, which have one task queue per thread and if one becomes empty, the thread is able to steal a task from a busy queue. Finally, I could narrow down the problem and found out that the most of the overall run time is spent in some leaf nodes, especially in the function `locate`, which retrieves the hit positions from the cursor.

As a consequence, I submit each invocation of `locate` to the thread pool, such that the search continues, while separate tasks deal with the creation of hits. This results in a large improvement of runtime and a nearly optimal thread usage. Furthermore, I have invented the criteria for omitting the `locate` step that I have described in item 4 on page 89, of which especially the pruning of very short queries has a large impact.

To summarize, the complete picture is this: For k stem loops there are k tasks submitted to the thread pool that recursively create search queries for a particular stem loop, compare lines 18 to 25 of listing 11.1. Each query that models a full stem loop and passes the filter submits another task to the pool for extracting the genome positions and storing the hits.

Since the search tasks in the pool generate new sub-tasks, it is desired that they are all submitted to the thread pool before the first sub-tasks are created. I have

```
1  struct ConcurrentFutureVector
2  {
3      std::vector<std::future<void>> futures;
4      std::mutex mutex;
5  };
6
7  void find_motif(BiDirectionalIndex const & index, Motif const & motif)
8  {
9      StemloopHitStore hits(index.get_names().size()); // allocate n vectors
10     uint8_t const num_motifs = motif.size(); // k
11     seqan3::detail::latch latch{num_motifs}; // initialize a latch
12
13     ConcurrentFutureVector locate_tasks;
14     std::vector<std::future<void>> search_tasks;
15
16     for (size_t idx = 0; idx < num_motifs; ++idx)
17     {
18         search_tasks.push_back(pool->submit([&index, &motif, &hits, &locate_tasks, &latch, idx]
19         {
20             // wait for latch
21             latch.wait();
22             // start recursive search
23             SearchInfo info(index.raw(), motif[idx], hits, locate_tasks);
24             recurse_search<LoopElement>(info, motif[idx].elements.cbegin(), 0);
25         }));
26         latch.arrive(); // decrement latch after submitting a task
27     }
28
29     for (auto & future: search_tasks) // first: wait for search tasks
30         future.wait();
31     for (auto & future: locate_tasks.futures) // second: wait for locate tasks
32         future.wait();
33
34     // ... work with hits
35 }
```

Listing 11.1: **Implementation of parallel motif search.** We use two vectors of futures to separate the search tasks from the locate sub-tasks and a latch to prioritize the search tasks. Lines 18 to 25 demonstrate that tasks can be submitted in the form of lambda functions.

implemented this with a *latch*: A counter starts from k (line 11) and is decremented with each submitted task (line 26). The tasks have a wait-command at their very beginning (line 21), which defers their start until the latch is released with the counter arriving to zero. Since the submission of k tasks (the loop in lines 16 to 27) usually takes only a few milliseconds, there is no noticeable delay in the program execution, but it is ensured that the first tasks do not already submit sub-tasks to the pool.

The futures of the tasks are collected in two separate vectors. The first one has length k and is used to wait for the stem loop iterations to finish (lines 29 to 30). The second one has an associated mutex (line 4), which controls that only one thread can append new futures for the `locate`-tasks at a time.

Since the hits are collected separately for each genome sequence, the multithreading of the final step is rather obvious: For each sequence `MaRs` submits a task to the thread pool, which performs the pooling of hits and collects motif matches.

12 Benchmarks

In order to demonstrate the performance of MaRs compared to relevant existing software, I have evaluated different benchmarks with focus on the search performance. In the following sections I investigate how well and with which demands of time and memory MaRs detects RNA families in genomic sequences compared to the tools Structator [Meyer et al., 2011] and Infernal [Nawrocki and Eddy, 2013]. This includes also evaluations with respect to the underlying index in the compressed and uncompressed case, like the construction time, the access time and the required disk space. In the last section of this chapter, I provide a comparison of the two output methods for matches in MaRs, which were discussed at the end of section 10.3.

For the benchmarks I have chosen from the best tools reported in recent publications, as analysed in section 2.3. The employed programs include Structator version 1.1, as found on GitHub[1], and Infernal version 1.1.4, which is available on its homepage[2]. The Structator tool is the most similar program to MaRs in the sense that it also employs a bi-directional index to search stem loops from inside to outside. Infernal is included as the currently leading tool for RNA homology search.

All benchmarks have been performed on a Linux server using an x86_64 architecture with Intel® Xeon® CPU E5-2650 v3 with 2.30 GHz and 126 GB RAM. I compiled with GCC version 9 and where applicable, I used up to 16 threads and AVX2 instructions.

Time and memory have been measured with the `/usr/bin/time -v` command, and I use `R` for plotting the graphs.

12.1 Comparison with Structator

In this section I describe the comparison of MaRs with Structator [Meyer et al., 2011]. I consider Structator the most similar tool, since it also employs a bi-directional index to match stem loops from inside to outside, i.e. starting with the hairpin loop. Differences exist however in the representations of index and motif. MaRs uses an FM-index based on Enhanced Prefixsum Rank dictionaries [Pockrandt et al., 2017], while Structator uses affix arrays as underlying index data structure. In contrast to MaRs, which defines the motif through detailed stem loop profiles as described in section 9.4, Structator takes pre-defined pattern descriptors as input.

[1] https://github.com/fernandomeyer/Structator (19.08.2022)
[2] http://eddylab.org/infernal/ (19.08.2022)

```
# index creation
afconstruct genome.fasta -alph dna.alphab -a -s genome
mars -g genome.fasta

# search
afsearch genome -comp dna_rna.comp -a -local -pat motifs.pat
mars -g genome.fasta -a msa.aln
```

Listing 12.1: **The commands for invoking Structator and MaRs.** The first pair of commands creates the indices, and the second one uses them to perform the search. The files `dna.alphab` and `dna_rna.comp` are shipped with `Structator`.

As genome input, I use the sequences compiled from the full alignments in the Rfam database [Griffiths-Jones et al., 2003], release 14.6. This set consists of 3,106,298 RNA sequences with a total length of 415 Megabases. I stored the sequences in a single *Fasta* file with the name `genome.fasta`, which is submitted to `MaRs` and `Structator` as shown in listing 12.1.

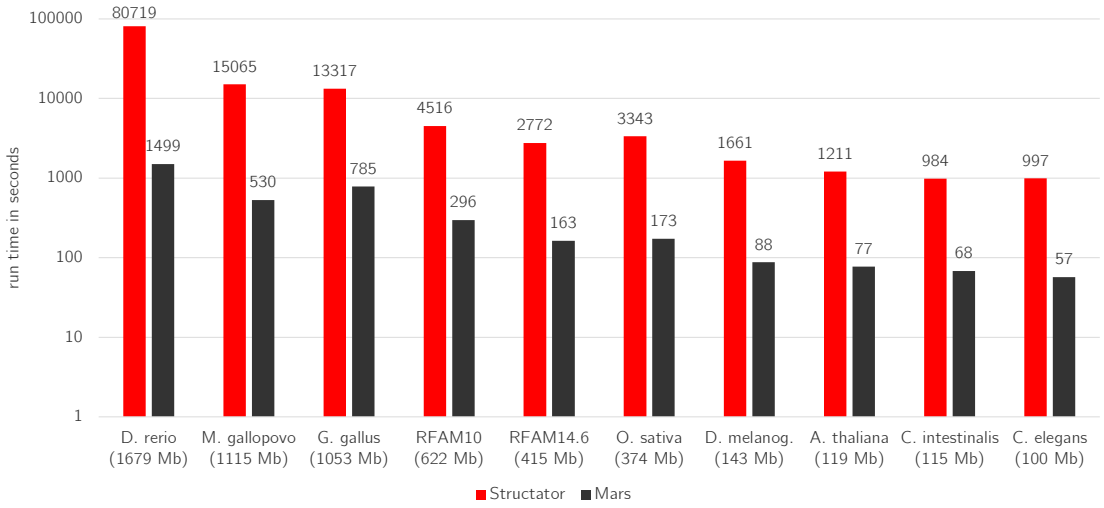
If `MaRs` only receives a genome file as input, it computes the index (if not already present) and terminates without creating motifs, since no alignment file is provided. However, please note that in contrast to `Structator`, `MaRs` is able to compute the index and perform the motif search in a single run, i.e. the first call of `MaRs` in listing 12.1 is not required in a normal workflow. For `Structator`, the `afconstruct` binary must be invoked, which takes besides the genome file also a file that defines the alphabet, and the `-a` flag, which tells the algorithm to compute all the tables of the index. This is the command as recommended in `Structator`'s readme-file.

Figure 12.1a shows the required time for creating indices for various genomes with `MaRs` and `Structator`. The comparison shows that `MaRs` is between 14 and 53× faster, depending on the genome size. Especially for large genomes `MaRs` convinces with a much more feasible index creation time: For the zebrafish genome (*Danio rerio*) of length 1679 Mb `MaRs` needs 25 minutes, whereas `Structator` takes 22.4 hours.

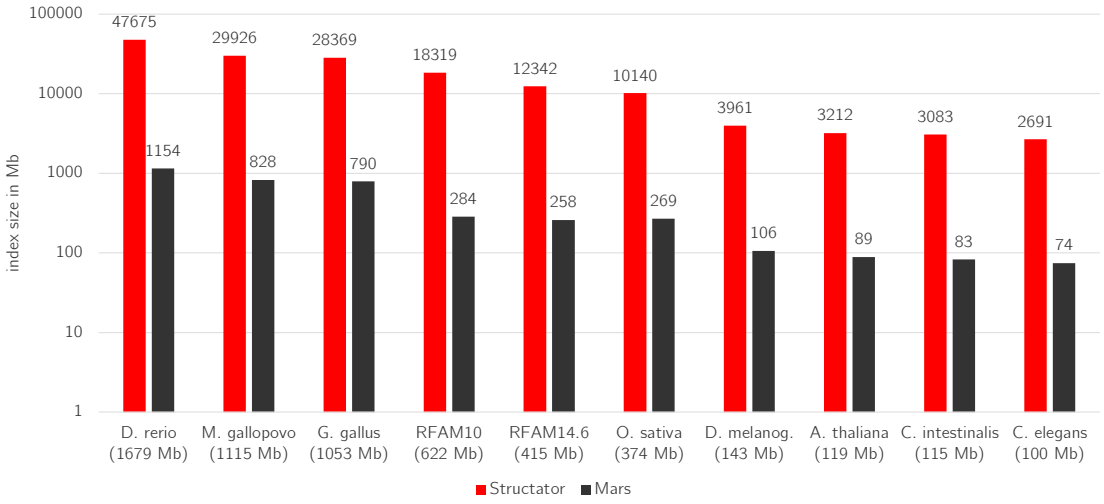
Regarding the size of the created index, `MaRs` is more efficient as well. As figure 12.1b demonstrates, the size of the index file on disk is between 36 and 64.5× larger for `Structator` compared to `MaRs`. Using the zebrafish example again, the index of `MaRs` has less than 1.2 gigabytes, while the files of `Structator` consume more than 47 gigabytes. I use the plural, because `Structator` creates 15 different files for a full index.

Note that in this benchmark `MaRs` creates the compressed index, as described in section 10.1 on page 86. The creation of the uncompressed index is about 25% faster compared to the compressed index, while its size is about 60% increased. For the zebrafish genome, I measured a run time of 18 minutes and 40 seconds and the resulting index size is 1.8 gigabytes.

The comparison of the search turned out to be difficult, since the input data is not compatible. While `MaRs` simply needs a multiple alignment as input, `Structator` reads a file with RNA sequence-structure patterns (RSSPs), which require a-priori



(a) Run time in relation to genome length.



(b) Index size in relation to genome length.

Figure 12.1: **Comparison of the index creation step.** I have computed indices for genomes of various length, and we can see that `MaRs` takes less time than `Structator` and creates a more compact index. For `MaRs` both benchmarks use the creation of the compressed index (`-z` enabled).

knowledge about the stem loops of the RNA family [Meyer et al., 2011]. Since RSSPs are similar to the stem loop descriptors in `MaRs`, I have implemented RSSP output in the `Stemloop` class, as shown in figure 9.3. The output is based on the fully-analysed stem loops and writes the `pat` format, for details I refer to section 9.4.

The executable to initiate the search with `Structator` is `afsearch`, which receives as parameters the base name of the genome index files (as specified in `afconstruct with the -s parameter`), a file `dna_rna.comp` that contains Watson-Crick and wobble complementarity rules, two flags `-a` and `-local`, since we have all tables available and want to search locally for matches, and the pattern file. The whole command is shown in listing 12.1. For `MaRs` the user must provide the genome filename and a multiple sequence-structure alignment (with or without secondary structure).

As alignment input, I took the seed alignments from Rfam that have at most 10 sequences. The use of deep alignments is excluded here, because they result in patterns with mainly wildcard characters. Since `Structator` patterns can only distinguish between nucleotides and wildcards without any weighting, it would be not fair to generate patterns from deep alignments. The benchmark set includes 2578 alignments, but for 187 of them `Structator` failed, so I excluded them such that this benchmark runs on 2391 RNA families. The multiple alignments are available as *Stockholm* files with secondary structure, however I computed also a second benchmark without secondary structures, so in this case they are computed with `IPknot`.

For the evaluation of an RNA family I compared the names of the detected sequences (the positive set) with the sequences that are assigned to this family in Rfam (the true set). From these sets I computed a confusion matrix, i.e. the true positives (tp), true negatives (tn), false positives (fp), and false negatives (fn). I calculated the values for the Matthews correlation coefficient (MCC), the sensitivity and the specificity according to equations (7.2) and (12.1). The average values over all 2391 RNA families are shown in table 12.1.

$$\text{sensitivity} := \frac{tp}{tp + fn} \qquad \text{specificity} := \frac{tn}{tn + fp} \tag{12.1}$$

The results show that `MaRs` and `Structator` both have excellent specificity, i.e. they do not report too many spurious results. However, we should keep in mind that we search in a set of 3,106,298 RNA sequences, and thus the number of true negatives is usually very high. The sensitivity of `MaRs` is almost double compared to `Structator`, which means that `MaRs` is better at detecting matches. In the benchmark with Rfam structures, `MaRs` reports on average 62% of the sequences that represent a family, while `Structator` detects only 35%.

The MCC values in table 12.1 reflect as well that `MaRs` is superior at detecting the correct matches for RNA families. The difference between the results computed with Rfam or `IPknot` structures is rather small; the Rfam structures lead to slightly better MCC values in this benchmark.

	MCC	sensitivity	specificity
MaRs	0.71	0.62	1.00
Structator	0.40	0.35	1.00

(a) structures from Rfam 14.6

	MCC	sensitivity	specificity
MaRs	0.65	0.60	1.00
Structator	0.36	0.37	0.99

(b) structures computed with IPknot

Table 12.1: **Evaluation of the motif matches.** The tables show the performance of MaRs and Structator for detecting RNA families. The consensus secondary structures are obtained either from Rfam 14.6, or computed with the IPknot program.

The run time for the search is visualized in figure 12.2 for Structator and MaRs, while for MaRs I show also the difference of using the compressed or uncompressed index. I want to express with this figure the distribution of the run times when computing the individual alignments. Therefore, I sorted the lists of run times in ascending order and plotted them to the same chart.

We can see that the vast majority of searches need very little time: for the lower 70% Structator takes 0.8 seconds per search, whereas MaRs takes 0.5 or 2.5 seconds per search for the lower 90% with the uncompressed and compressed index, respectively. In 99% of the cases, both programs need less than 20 seconds for a search (in absolute numbers, 2362 of the searches finished below 20 seconds). At the upper end there are a few cases where MaRs takes more time than Structator. The extreme case with a run time of 109 minutes is the family RF00503, where IPknot detects a very complex secondary structure, which has many rather small stem loops that generate thousands of hits.

What is also clearly visible from figure 12.2 is that decompressing the index for a search takes constantly 2 seconds, using the index of the Rfam 14.6 sequences. My general recommendation is not to use compression, especially if many searches are being performed. The run time difference in this benchmark with 2391 searches in total is about 80 minutes. However, if the disk space is too limited for storing large indices, the compression is a vital feature of MaRs with still very acceptable run times.

The conclusion about this benchmark is that MaRs provides much better results in detecting genome locations than Structator, while it is faster in most of the cases. In contrast to Structator, MaRs does not need pre-defined patterns as input and thus can be used more easily in automated workflows. Furthermore, MaRs requires substantially less memory for the index, which makes it amenable for applying huge amounts of data.

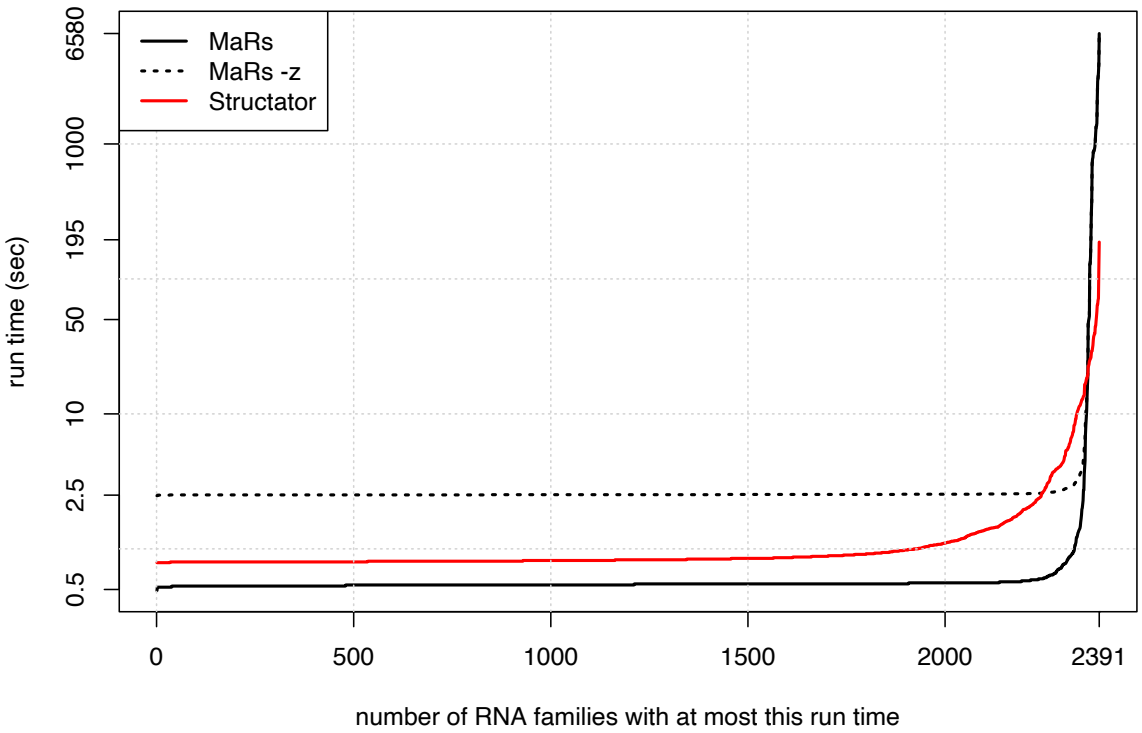


Figure 12.2: **Run time of motif searches with MaRs and Structator.** The plot shows the run time values sorted in ascending order, so the distribution of rather fast and slow searches is visible. For `MaRs` I included two curves for comparing the use of a compressed (`-z`) and uncompressed index.

12.2 RMARK

`Infernal` uses rather different methods compared to `MaRs` and `Structator`. It does not use an index for searching in sub-linear time, but scans the genome sequences in a dynamic programming style. Instead of motifs or patterns it employs covariance models, i.e. consensus structure profiles that define an RNA family. For more background about `Infernal` I refer to section 2.3.

For a comparison with `Infernal`, my first attempt was using the `RMARK3` benchmark scripts, that are available on GitHub in the `Infernal` repository[3]. The scripts are written in Perl and have been developed by Nawrocki and Eddy, 2013 based on Rfam version 10. Besides `Infernal`, they include also `nhmmer` [Wheeler and Eddy, 2013] and `BLASTN` [Altschul et al., 1997] to the tests.

This benchmark runs on the seed alignments of 106 RNA families with at least 5 aligned sequences. The genome sequence set of length 10.16 Mb is created from random sequences, in which 780 test sequences are inserted that have less than 60% identity with the alignment sequences [Nawrocki and Eddy, 2013].

[3] `https://github.com/EddyRivasLab/infernal` (22.08.2022)

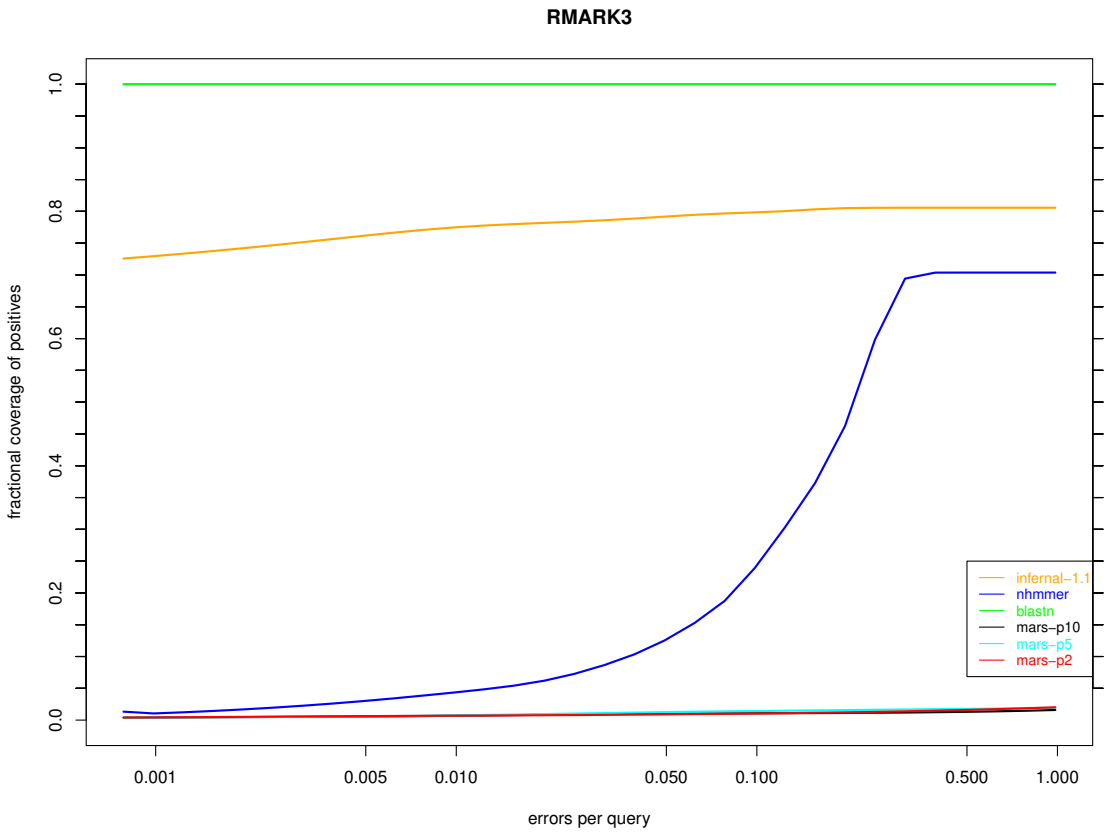


Figure 12.3: **Result of the RMARK3 benchmark.** The plot shows the detection rate of sequences (sensitivity) versus the false-positive rate for the tested programs.

I have added `MaRs` to the benchmark by implementing an additional Perl script to be executed by `RMARK3`. `MaRs` is tested three times with the profile pruning parameter p set to 10, 5, and 2 (compare section 9.4.3).

As we can see in figure 12.3, `MaRs` detects only less than 10% of the sequences in this benchmark, whereas `BLASTN` shows perfect sensitivity and `Infernal` performs between 70 and 80%. Since the sequences differ so much from the alignment sequences by benchmark design, `MaRs` is unable to detect many of them. `MaRs` currently employs an exact search, i.e. the search queries are derived directly from the sequence-structure profile. Nucleotides that are absent in an alignment column are not considered for the search.

`MaRs` could be extended in a future version, such that it accepts up to a particular amount of errors in the search, where error is meant in the sense of an edit operation. However, the ratio of errors will always remain rather small, since for each error all the possible nucleotides or nucleotide pairs must be considered, which leads quickly to a combinatorial explosion in the search tree: allowing for k errors creates 4^k branches in a loop region and 24^k branches in a stem region (compare section 9.4.3).

```
cmbuild model.cm msa.sth
cmcalibrate model.cm
cmsearch model.cm genome.fasta
```

Listing 12.2: **The commands for invoking Infernal.** The first step reads an alignment and computes a CM, which needs to be calibrated using the second command. The final step performs the search in the genome.

Additionally, a method for allowing additional gaps of various lengths to some extent should then be developed as well.

12.3 Comparison with Infernal

Two more realistic benchmarks for `MaRs` are the search for RNA families in all the sequences given in Rfam, i.e. without explicitly excluding all the similar ones, or in a real genome sequence, e.g. the human genome. Let us focus on the first case here, and I cover the search within a human genome in the following section.

In this benchmark I want to compare the performance of `Infernal`, `Structator`, and `MaRs`, where for `MaRs` I use two configurations, the default $p = 10$ and a more sensitive one with $p = 2$. As sequence data I have used the genomic sequences annotated in Rfam 14.6, which consist of 3,106,298 RNA sequences with a total length of 415 Megabases. Furthermore, I have downloaded the complete set of seed alignments from Rfam, i.e. the alignments of 4070 families in *Stockholm* format.

When creating the pattern files for `Structator`, I realized that 388 families do not have base pair interactions in their secondary structure string, e.g. different snoRNAs. Since in this case `MaRs` cannot find any stem loops and thus there are no RSSPs to write to a pattern file, I excluded these RNA families from the benchmark. Later on, I had to exclude additional 3 families (`RF02540`, `RF02541`, `RF02543`), because the `cmsearch` command of `Infernal` did not terminate successfully. Interestingly, these are the three families with the longest alignments. In the end, the benchmark set consists of 3722 families.

We have already seen in listing 12.1 how `MaRs` and `Structator` are executed. Note that for `MaRs` we add a call with the `-p 2` setting to test also a more relaxed profile filter. Listing 12.2 shows the commands for invoking `Infernal`, which involve the model creation, the calibration, and the actual search.

I have run the programs as described, and for each family the reported sequences are compared with the set of assigned sequences in Rfam. This comparison generates a confusion matrix, of which the sensitivity, specificity, and MCC are computed according to equations (7.2) and (12.1). The results are shown in figure 12.4 and table 12.2.

For the plot I use a lowess smoother ($f = \frac{2}{3}$) to generate the curves from the data points for each family. Since the specificity is very high, I decided to plot the values for $1 -$ specificity, i.e. the false positive rate, on the logarithmic x-axis. It shows

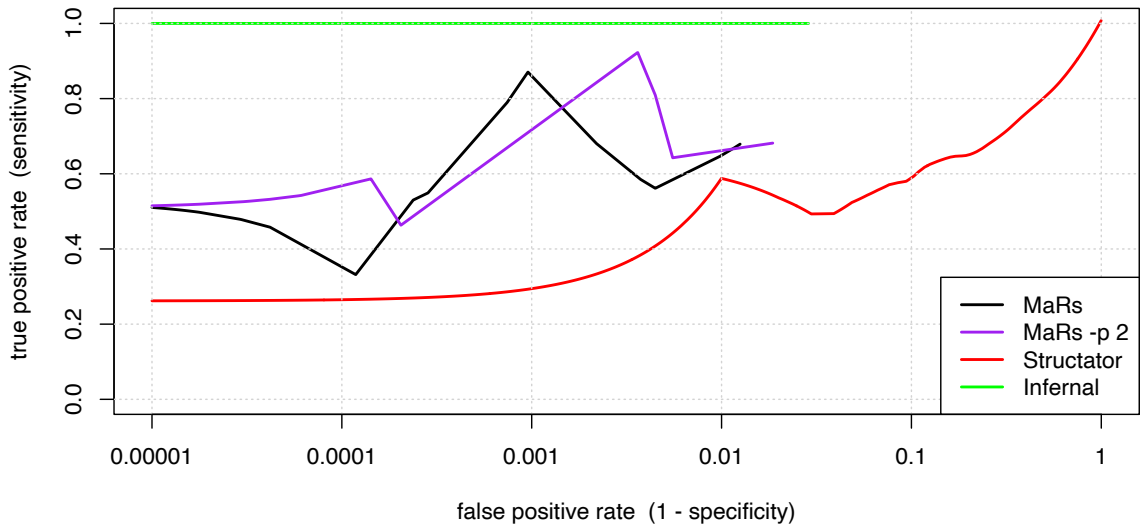


Figure 12.4: **Comparison of the performance for MaRs, Structator, and Infernal.** The plot shows the relation of true positive rate and false positive rate for the tested programs. The x-axis is logarithmic and all values have been shifted by $\varepsilon = 10^{-5}$ to avoid zeroes. The curves end at the data point with the maximal false positive rate.

	MCC	sensitivity	specificity
`MaRs`	0.654	0.509	0.999984
`MaRs-p 2`	0.659	0.513	0.999991
`Structator`	0.319	0.379	0.971228
`Infernal`	0.919	0.995	0.999956

Table 12.2: **Results of the Infernal benchmark.** The table shows the average values of MCC, sensitivity, and specificity for detecting RNA families.

how likely a program reports a spurious match. On the y-axis is shown how likely a program detects a valid match (sensitivity, true positive rate).

`Infernal` detects on average 99.5% of the sequences that belong to the RNA families, `MaRs` in both versions around 51%, and `Structator` 38%. The high sensitivity of `Infernal` can be explained with the fact that the annotations in the Rfam database that we are using have been created with the `Infernal` tool.

`MaRs` misses the sequences with content that is not represented in the alignments. At the same time `MaRs` has the best specificity among the tested tools, i.e. it does not report many wrong matches. Likely, we can relax the output filter, such that more matches are generated. This would increase the sensitivity, but decrease specificity. The benchmark in section 12.5 demonstrates how the results of a comparably weak filter look like.

`Structator` has a much lower specificity compared to the benchmark in section 12.1, compare table 12.1. Since this benchmark contains also deep alignments,

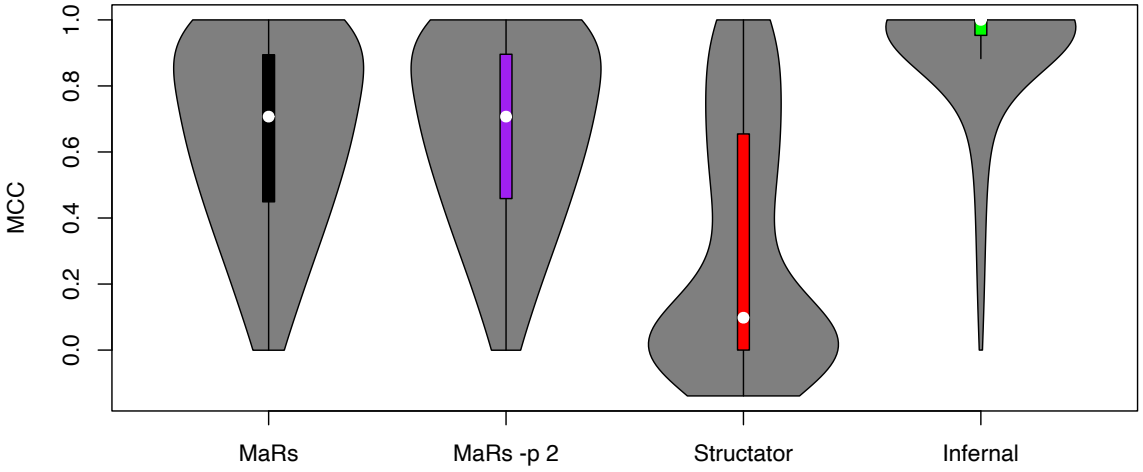


Figure 12.5: **MCC distributions in the Infernal benchmark.** The white dot marks the median value, and the coloured bars end at the quartiles.

i.e. alignments with many sequences, the most alignment columns generate the wildcard `N` for the pattern descriptor. Thus, `Structator` searches for more unspecific patterns in this case, and coincidently finds more true and false matches.

The Matthews Correlation Coefficient (MCC) is a balanced measure for the correlation of the actual and the predicted sequence sets. The distribution of the MCC is visualized in the violin plots in figure 12.5 and the results coincide with the analysis of the previous paragraphs.

Comparing the run time in figure 12.6 demonstrates that `MaRs` with default parameters is the fastest tool among the programs. Compared to `Infernal` it is faster by orders of magnitude. I have also added the run times of `MaRs` with single-thread execution (`-j 1`) to the plot. While `MaRs` benefits from a very efficient and highly parallel implementation and an index with sub-linear search, `Infernal` suffers from the expensive creation and calibration of the covariance model. `Structator` is also rather fast, since it also benefits from a bi-directional index. Its run time is between the run times of the single- and multithreaded `MaRs` configurations.

Furthermore, the plot shows the influence of the profile pruning on run time. The more sensitive search takes significantly more time compared to the default $p = 10$ setting.

`MaRs` has the lowest consumption of working memory, as it is clearly visible from figure 12.7. However, for longer alignments there is a big difference between the default and the $p = 2$ configuration, since the more sensitive run needs to store more hits. For short alignments, `Structator` allocates the most working memory compared to the other programs, this effect is probably caused by the index, since `Structator`'s memory increases only for longer alignments. `Infernal` shows a nearly linear correlation of alignment length and memory consumption, and for long alignments it consumes more memory than `Structator`.

Since `MaRs` is so fast compared to `Infernal`, I wanted to find out for how many

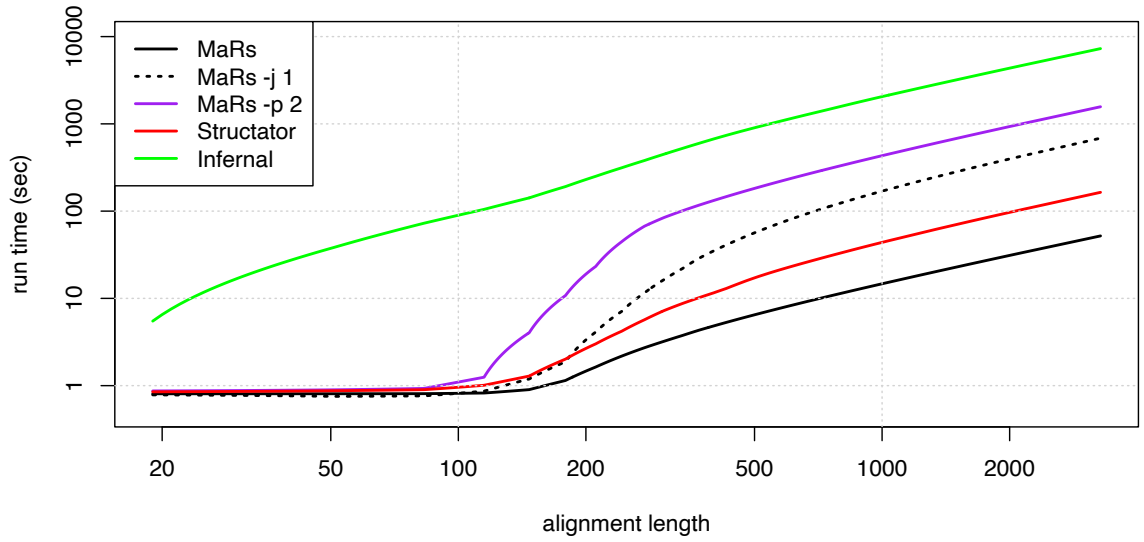


Figure 12.6: **Runtime comparison in the Infernal benchmark.** For `MaRs` the choice of the pruning parameter and single-threaded execution has a large impact on the run time, but in its default version `MaRs` is the fastest tool. `Infernal` is clearly slower on the whole range of alignment lengths.

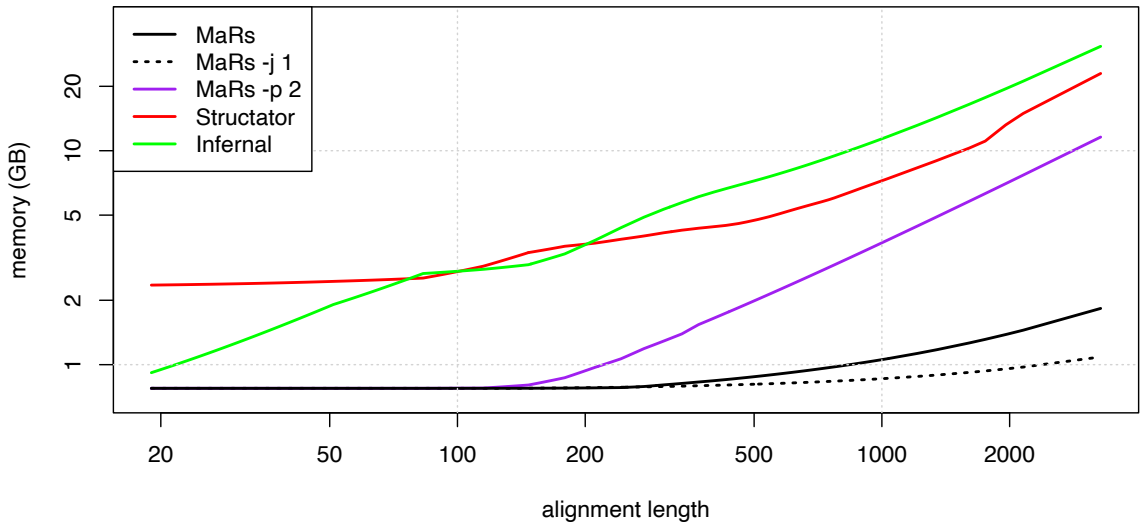


Figure 12.7: **Memory comparison in the Infernal benchmark.** For all the programs, memory increases with alignment length. We can see that `MaRs` has the lowest memory footprint and that the parameter p for pruning the motif has a big impact. I have added a memory curve for single-threaded execution of `MaRs` (`-j 1`).

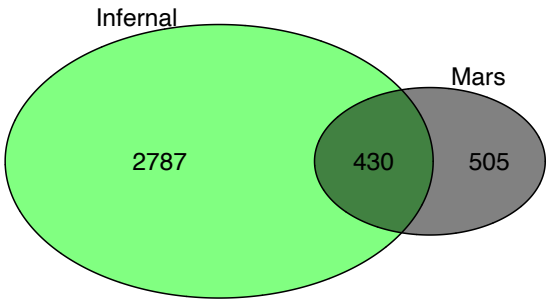


Figure 12.8: **Direct comparison of RNA family detection.** I have computed for how many RNA families `MaRs` or `Infernal` is superior according to MCC. For 935 families `MaRs` is at least as good as `Infernal`. For 930 of them `MaRs` computes them in a shorter time.

RNA families it is actually more beneficial to use `MaRs`. The direct comparison per family according to the MCC criterion is shown in the Venn diagram in figure 12.8. We can see from the numbers that for 935 families the MCC of `MaRs` is at least as large as the MCC of `Infernal`. For 930 of these families `MaRs` computes the matches faster. The median run time per alignment with `Infernal` is 89.14 seconds, with `MaRs` it is 0.57 seconds.

12.4 Search in the Human Genome

In this section I briefly want to describe another benchmark I have set up with `MaRs` and `Infernal`, based on the human genome. I have downloaded the genome version *GRCh38.p11* with a size of 3.25 Gb. Furthermore, I have extracted from Rfam the annotated RNA families and got the alignments of 1066 families.

`MaRs` created an index of the genome in 22 minutes and took additional 5 hours and 38 minutes for finding the 1066 RNA families with help of the index. I verified for each family the locations, by comparing the annotation with the positions reported by `MaRs`. On average, `MaRs` has a sensitivity of 0.45, i.e. it identified 45% of each family's occurrences.

However, if we change the viewpoint away from the individual RNA families, in total there are 14,910 locations annotated in Rfam, of which `MaRs` finds 923 i.e. only 6.2% of them. My conclusion from these two different results is that `MaRs` works well for families that have a few well-conserved transcripts in the genome, while for other families (apparently the ones with a lot of transcripts) it does not.

Unfortunately, `Infernal` could not finish this benchmark. When I checked the status after 72 hours run time, it had only processed 10 of the 25 chromosome sequences.

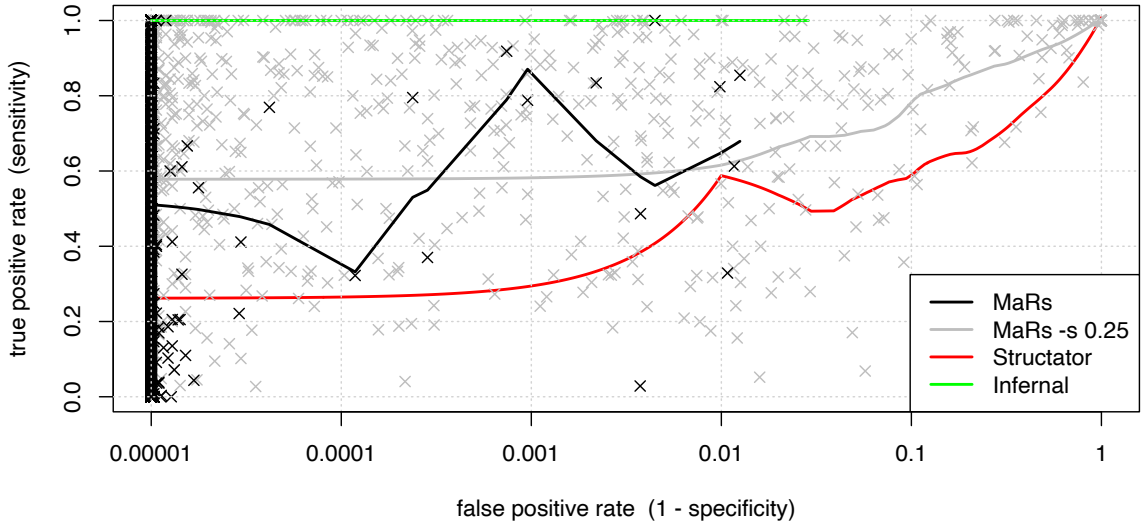


Figure 12.9: **Comparison of the output filter setting.** This plot is created like figure 12.4, but it adds a curve for a different output filter and shows the data points of `MaRs`. The black curve uses the default e-value criterion, while the grey curve requires a minimum score of $s = 0.25$ per stem loop contained in the motif.

12.5 Comparison of the output filter methods

At the end of section 10.3 I have described two different methods for filtering the best motif matches. The default method, which is displayed in solid black in figures 12.2 to 12.7, uses a threshold of e-values and produces results of very high specificity. The other method requires that more than one quarter of the stem loops must be present in the match and the score per stem loop must be larger than s, which is 0.25 in this benchmark.

What we see from figure 12.9 is that the specificity is much lower with the score filter: While in the default method almost all the data points are at the left border, the grey data points scatter over the whole range. In numbers, the specificity of the score filter is 0.98760 compared to 0.99999 of the e-value method (compare table 12.2). The amount of false positives has also dropped the MCC value to 0.644 for the score filter.

Beneficial is however the improvement in sensitivity, which is on average 0.60 with the score filter and 0.51 with the e-values. Thus, `MaRs` is able to detect more matches, and it may be worth investing further efforts into a fine-tuning of this method.

As a conclusion, with setting $s = 0.25$ `MaRs` finds generally more matches in the genome, which leads to an improved sensitivity and a reduced specificity.

13 Discussion

In part III of this thesis I have described my implementation of a new C++ program with the name `MaRs`, which is the acronym for Motif-based aligned RNA searcher. It is capable of rapidly finding the transcripts of an RNA family in a genomic sequence or database, given a multiple sequence-structure alignment.

I consider `MaRs` a valuable contribution to the field of RNA homology search. `MaRs` is able to model pseudoknots in contrast to many of the existing tools, including the SCFG-based tools like `Infernal`. We have seen not only that `MaRs` is extremely fast and capable of searching the whole human genome for over 1000 RNA families in just six hours, it is also very economic in memory allocation. Thus, `MaRs` is well-prepared for analysing enormous amounts of data.

The speed-up of `MaRs` compared to `Infernal` is on average 159×, and in the maximum case 7511×. The enormous speed of `MaRs` is based on the interplay of (a) the application of the efficient bi-directional FM-index from `SeqAn 3`, (b) the idea of searching stem loops from inside to outside, which has been borrowed from `Structator`, (c) intelligent pruning of the search paths, and (d) the implementation of multithreading techniques. As described in chapter 11, I have put much effort into optimizing the code with the help of a thread pool, such that the available CPU cores are used as efficient as possible.

The benchmarks in the previous chapter demonstrate as well that `MaRs` needs some additional tuning to increase the sensitivity. I believe that `MaRs` would benefit the most from well-designed methods for accepting a limited amount of errors during the search. I mean errors in the sense of base substitutions, insertions, or deletions, so the algorithm is not any more limited to the options in the alignment. These methods must be designed with care, since they counteract the profile pruning, and we have seen in the benchmarks for $p = 2$ that the run time depends much on the number of search paths.

`SeqAn 3` [Reinert et al., 2017] played an important role and was a big help for implementing `MaRs`, since many useful algorithms and data structures are provided, like alphabets with wildcard or gap support, sequence file input, the gzip stream adaptor, the bidirectional FM-index, and the wrappers for some C++ 20 features like latch and zip-iterator.

Finally, I would like to stress that with `MaRs` I have implemented a very user-friendly software, which provides various features. It implements different methods for filtering the found matches, it is capable of reading and writing a possibly compressed index, it can store and read the motif to/from a file, and convert the motif into a pattern file for `Structator`. Furthermore, `MaRs` employs a thread pool with a selectable number of threads, and as input it accepts versatile formats for sequences and alignments, which may or may not contain secondary structure.

13.1 Outlook

In this section I would like to suggest various directions for the future development of the `MaRs` tool. As stated a few paragraphs above, `MaRs` needs a sophisticated scheme for considering nucleotides or gaps to a small extent in the search that are not represented in the alignment. I suggest that the inclusion of one or two edit operations in the search already enables many more hits to be found. However, if `MaRs` should gain the ability of finding matches in very distant sequences like in the RMARK benchmark (section 12.2), this is likely not enough and in-depth research is needed to find efficient methods that allow for more variability.

For improving the balance of sensitivity and specificity for the resulting matches further, the two presented methods should be fine-tuned regarding the e-value threshold or the score criterion. Especially a good value for s is hard to find, since I found that the optimal value depends on the observed alignment and varies among different RNA families, although I already account for the number of stem loops. Furthermore, the pooling of hits can be improved from a simple iteration to a more sophisticated chaining, like in `Structator`. This would likely result in better groups of hits and thus in a clearer signal that allows to differentiate high-scoring from spurious matches.

Regarding the benchmarks, the output of pattern files by `MaRs` caused the problem of many wildcard symbols for deep alignments. To provide more specific input files for `Structator` I suggest setting a nucleotide if an alignment column consists of at least 80% or 90% of the same nucleotide. The current threshold of 100% seems too strict. Additionally, I realized that all my benchmarks depend more or less on Rfam, the resource for RNA families. Since `Infernal` is involved in creating this database, it would be interesting to see how the programs compare based on different resources.

The methods for computing the alignment secondary structure could be complemented with the tool `SimulFold` [Meyer and Miklós, 2007], which is a promising approach for inferring the RNA structure. It performs Bayesian inference with Markov chain Monte Carlo and shows good results compared to `RNAalifold`. It is capable of predicting pseudoknots, however the performance on the pseudoknotted samples in the publication have a comparably low MCC result with `SimulFold`. A benchmark by Doose and Metzler, 2012 shows superior results for `SimulFold` compared to `IPknot`, but even better results for their `PhyloQFold` tool.

13.2 Availability of MaRs

Source code: freely available on `https://github.com/seqan/mars`
System requirements: tested on Linux and MacOS with gcc ≥ 7
Software dependencies: `SeqAn` 3.1, `IPknot` 1.0.0
License: BSD-License (3-clause)

Part IV

Conclusion and back matter

14 Conclusion

In this thesis I have presented my contributions to the field of computational RNA analysis. We have seen that for RNA molecules the analysis of the sequence alone is not enough, because rather the secondary structure is evolutionary conserved. In order to extract the conserved motifs from a set of homologous RNA sequences, I have implemented the LaRA 2 program. It computes a sequence-structure alignment from the provided set of homologous RNA sequences. The consideration of secondary structure in an alignment, especially if pseudoknots should be considered, adds so much complexity to the alignment problem that new and faster programs are needed to cope with the vast amount of data.

LaRA 2 is a C++ program that is capable of computing structural alignments with pseudoknots in high quality. I have implemented new efficient alignment and matching functions, and used multithreading and vectorization techniques to make LaRA 2 up to 130× faster than its predecessor LaRA 1. For computing multiple structural alignments, I provide two workflows that use MAFFT or T-Coffee for progressively combining pairwise alignments.

The extraction the conserved structural motifs from the multiple structural alignments is one of the tasks for which I have developed my second tool: MaRs. I have designed sophisticated descriptors that store the relevant information in the form of stem loops. The stem loop descriptors usually characterize an RNA family, and they can be utilized to find homologs in a genome sequence. These are the positions of the genome where further members of the same RNA family are encoded.

MaRs can rapidly detect matching genome locations from the stem loop profiles. For the search I employ a bi-directional index data structure, which allows performing searches in sub-linear time and extending the search pattern in both directions, as it is the nature of stem loops. I have implemented MaRs as a C++ program, which benefits from extensive multithreading and effective pruning techniques.

The development of the presented workflow was a very interesting and challenging PhD project for me. I really enjoyed bringing LaRA 2 and MaRs into life, after designing them from scratch. I have learned much about parallel and vectorized programming in a modern C++ environment, and got many insights into RNA biology. As part of the SeqAn team I was happy to contribute to various library features and have always enjoyed the good team work in our group.

Acknowledgements

First of all, I would like to thank *Knut Reinert* for being my mentor, and I am very grateful that he has guided me through my PhD project. After each discussion we have had about my project, I have been equipped not only with new solutions and many ideas, but also with a boost of motivation.

Furthermore, I want to thank *Annalisa Marsico* and *Martin Vingron*, who were members of my thesis advisory committee and supported me with their experience and helpful advice.

Gianvito Urgese has always been a great source of motivation and ideas for me. He supported me very much with the LaRA 2 project and invited me to Torino, where we worked together for a week and had many fruitful discussions.

I have appreciated very much working in the *SeqAn team*. I could learn and discover so much from and with the members of the group, including agile project management techniques and how to design a modern C++ software library. Retreats, sports activities and many tea breaks with the group members have made the work days pleasant and fun.

Finally, I would like to express my gratitude to the *IMPRS*, especially to *Kirsten Kelleher* and *Anne-Dominique Gindrat*, who did an amazing job as PhD coordinators at the Max Planck Institute for Molecular Genetics. I always felt very supported there and during the various events I could connect with renowned scientists as well as peers, of which some have become good friends.

Bibliography

[Altschul et al., 1997] Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. J. (1997). Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Res*, 25(17):3389–402.

[Athanasopoulos et al., 1999] Athanasopoulos, V., Praszkier, J., and Pittard, A. J. (1999). Analysis of elements involved in pseudoknot-dependent expression and regulation of the repa gene of an incl/m plasmid. *J Bacteriol*, 181(6):1811–9.

[Bartel, 2009] Bartel, D. P. (2009). Micrornas: target recognition and regulatory functions. *Cell*, 136(2):215–33.

[Bauer and Klau, 2005] Bauer, M. and Klau, G. W. (2005). Structural alignment of two rna sequences with lagrangian relaxation. In Fleischer, R. and Trippen, G., editors, *Algorithms and Computation*, pages 113–123, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Bauer et al., 2007] Bauer, M., Klau, G. W., and Reinert, K. (2007). Accurate multiple sequence-structure alignment of rna sequences using combinatorial optimization. *BMC Bioinformatics*, 8(1).

[Bauer et al., 2008] Bauer, M., Klau, G. W., and Reinert, K. (2008). An exact mathematical programming approach to multiple RNA sequence-structure alignment. *Algorithmic Oper. Res.*, 3(2).

[Bayegan and Clote, 2020] Bayegan, A. H. and Clote, P. (2020). Rnamountalign: Efficient software for local, global, semiglobal pairwise and multiple rna sequence/structure alignment. *PLoS One*, 15(1):e0227177.

[Berman et al., 1992] Berman, H. M., Olson, W. K., Beveridge, D. L., Westbrook, J., Gelbin, A., Demeny, T., Hsieh, S. H., Srinivasan, A. R., and Schneider, B. (1992). The nucleic acid database. a comprehensive relational database of three-dimensional structures of nucleic acids. *Biophys J*, 63(3):751–9.

[Capriotti and Marti-Renom, 2010] Capriotti, E. and Marti-Renom, M. A. (2010). Quantifying the relationship between sequence and three-dimensional structure conservation in rna. *BMC Bioinformatics*, 11:322.

[Chan and Lowe, 2019] Chan, P. P. and Lowe, T. M. (2019). tRNAscan-SE: Searching for tRNA genes in genomic sequences. *Methods Mol Biol*, 1962:1–14.

[Chatzou et al., 2016] Chatzou, M., Magis, C., Chang, J.-M., Kemena, C., Bussotti, G., Erb, I., and Notredame, C. (2016). Multiple sequence alignment modeling: methods and applications. *Brief Bioinform*, 17(6):1009–1023.

[Cock et al., 2009] Cock, P. J. A., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., and de Hoon, M. J. L. (2009). Biopython: freely available python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–3.

[Constanty and Shkumatava, 2021] Constanty, F. and Shkumatava, A. (2021). lncrnas in development and differentiation: from sequence motifs to functional characterization. *Development*, 148(1).

[Cox et al., 2000] Cox, D. N., Chao, A., and Lin, H. (2000). piwi encodes a nucleoplasmic factor whose activity modulates the number and division rate of germline stem cells. *Development*, 127(3):503–14.

[Crick, 1970] Crick, F. (1970). Central dogma of molecular biology. *Nature*, 227(5258):561–3.

[Crick, 1958] Crick, F. H. (1958). On protein synthesis. *Symp Soc Exp Biol*, 12:138–63.

[Daily, 2016] Daily, J. (2016). Parasail: Simd c library for global, semi-global, and local pairwise sequence alignments. *BMC Bioinformatics*, 17:81.

[Danaee et al., 2018] Danaee, P., Rouches, M., Wiley, M., Deng, D., Huang, L., and Hendrix, D. (2018). bprna: large-scale automated annotation and analysis of rna secondary structure. *Nucleic Acids Research*, 46(11):5381–5394.

[Dezső et al., 2011] Dezső, B., Jüttner, A., and Kovács, P. (2011). Lemon – an open source c++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23–45.

[Dirks and Pierce, 2004] Dirks, R. M. and Pierce, N. A. (2004). An algorithm for computing nucleic acid base-pairing probabilities including pseudoknots. *J Comput Chem*, 25(10):1295–304.

[Do et al., 2006] Do, C. B., Woods, D. A., and Batzoglou, S. (2006). Contrafold: Rna secondary structure prediction without physics-based models. *Bioinformatics*, 22(14):e90–8.

[Doose and Metzler, 2012] Doose, G. and Metzler, D. (2012). Bayesian sampling of evolutionarily conserved RNA secondary structures with pseudoknots. *Bioinformatics*, 28(17):2242–2248.

[Edmonds, 1965a] Edmonds, J. (1965a). Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56.

[Edmonds, 1965b] Edmonds, J. (1965b). Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467.

[ENCODE Project Consortium et al., 2007] ENCODE Project Consortium, Birney, E., Stamatoyannopoulos, J. A., Dutta, A., Guigó, R., Gingeras, T. R., Margulies, E. H., Weng, Z., Snyder, M., Dermitzakis, E. T., Thurman, R. E., Kuehn, M. S., Taylor, C. M., Neph, S., Koch, C. M., Asthana, S., Malhotra, A., Adzhubei, I., Greenbaum, J. A., Andrews, R. M., Flicek, P., Boyle, P. J., Cao, H., Carter, N. P., Clelland, G. K., Davis, S., Day, N., Dhami, P., Dillon, S. C., Dorschner, M. O., Fiegler, H., Giresi, P. G., Goldy, J., Hawrylycz, M., Haydock, A., Humbert, R., James, K. D., Johnson, B. E., Johnson, E. M., Frum, T. T., Rosenzweig, E. R., Karnani, N., Lee, K., Lefebvre, G. C., Navas, P. A., Neri, F., Parker, S. C. J., Sabo, P. J., Sandstrom, R., Shafer, A., Vetrie, D., Weaver, M., Wilcox, S., Yu, M., Collins, F. S., Dekker, J., Lieb, J. D., Tullius, T. D., Crawford, G. E., Sunyaev, S., Noble, W. S., Dunham, I., Denoeud, F., Reymond, A., Kapranov, P., Rozowsky, J., Zheng, D., Castelo, R., Frankish, A., Harrow, J., Ghosh, S., Sandelin, A., Hofacker, I. L., Baertsch, R., Keefe, D., Dike, S., Cheng, J., Hirsch, H. A., Sekinger, E. A., Lagarde, J., Abril, J. F., Shahab, A., Flamm, C., Fried, C., Hackermüller, J., Hertel, J., Lindemeyer, M., Missal, K., Tanzer, A., Washietl, S., Korbel, J., Emanuelsson, O., Pedersen, J. S., Holroyd, N., Taylor, R., Swarbreck, D., Matthews, N., Dickson, M. C., Thomas, D. J., Weirauch, M. T., Gilbert, J., Drenkow, J., Bell, I., Zhao, X., Srinivasan, K. G., Sung, W.-K., Ooi, H. S., Chiu, K. P., Foissac, S., Alioto, T., Brent, M., Pachter, L., Tress, M. L., Valencia, A., Choo, S. W., Choo, C. Y., Ucla, C., Manzano, C., Wyss, C., Cheung, E., Clark, T. G., Brown, J. B., Ganesh, M., Patel, S., Tammana, H., Chrast, J., Henrichsen, C. N., Kai, C., Kawai, J., Nagalakshmi, U., Wu, J., Lian, Z., Lian, J., Newburger, P., Zhang, X., Bickel, P., Mattick, J. S., Carninci, P., Hayashizaki, Y., Weissman, S., Hubbard, T., Myers, R. M., Rogers, J., Stadler, P. F., Lowe, T. M., Wei, C.-L., Ruan, Y., Struhl, K., Gerstein, M., Antonarakis, S. E., Fu, Y., Green, E. D., Karaöz, U., Siepel, A., Taylor, J., Liefer, L. A., Wetterstrand, K. A., Good, P. J., Feingold, E. A., Guyer, M. S., Cooper, G. M., Asimenos, G., Dewey, C. N., Hou, M., Nikolaev, S., Montoya-Burgos, J. I., Löytynoja, A., Whelan, S., Pardi, F., Massingham, T., Huang, H., Zhang, N. R., Holmes, I., Mullikin, J. C., Ureta-Vidal, A., Paten, B., Seringhaus, M., Church, D., Rosenbloom, K., Kent, W. J., Stone, E. A., NISC Comparative Sequencing Program, Baylor College of Medicine Human Genome Sequencing Center, Washington University Genome Sequencing Center, Broad Institute, Children's Hospital Oakland Research Institute, Batzoglou, S., Goldman, N., Hardison, R. C., Haussler, D., Miller, W., Sidow, A., Trinklein, N. D., Zhang, Z. D., Barrera, L., Stuart, R., King, D. C., Ameur, A., Enroth, S., Bieda, M. C., Kim, J., Bhinge, A. A., Jiang, N., Liu, J., Yao, F., Vega, V. B.,

Lee, C. W. H., Ng, P., Shahab, A., Yang, A., Moqtaderi, Z., Zhu, Z., Xu, X., Squazzo, S., Oberley, M. J., Inman, D., Singer, M. A., Richmond, T. A., Munn, K. J., Rada-Iglesias, A., Wallerman, O., Komorowski, J., Fowler, J. C., Couttet, P., Bruce, A. W., Dovey, O. M., Ellis, P. D., Langford, C. F., Nix, D. A., Euskirchen, G., Hartman, S., Urban, A. E., Kraus, P., Van Calcar, S., Heintzman, N., Kim, T. H., Wang, K., Qu, C., Hon, G., Luna, R., Glass, C. K., Rosenfeld, M. G., Aldred, S. F., Cooper, S. J., Halees, A., Lin, J. M., Shulha, H. P., Zhang, X., Xu, M., Haidar, J. N. S., Yu, Y., Ruan, Y., Iyer, V. R., Green, R. D., Wadelius, C., Farnham, P. J., Ren, B., Harte, R. A., Hinrichs, A. S., Trumbower, H., Clawson, H., Hillman-Jackson, J., Zweig, A. S., Smith, K., Thakkapallayil, A., Barber, G., Kuhn, R. M., Karolchik, D., Armengol, L., Bird, C. P., de Bakker, P. I. W., Kern, A. D., Lopez-Bigas, N., Martin, J. D., Stranger, B. E., Woodroffe, A., Davydov, E., Dimas, A., Eyras, E., Hallgrímsdóttir, I. B., Huppert, J., Zody, M. C., Abecasis, G. R., Estivill, X., Bouffard, G. G., Guan, X., Hansen, N. F., Idol, J. R., Maduro, V. V. B., Maskeri, B., McDowell, J. C., Park, M., Thomas, P. J., Young, A. C., Blakesley, R. W., Muzny, D. M., Sodergren, E., Wheeler, D. A., Worley, K. C., Jiang, H., Weinstock, G. M., Gibbs, R. A., Graves, T., Fulton, R., Mardis, E. R., Wilson, R. K., Clamp, M., Cuff, J., Gnerre, S., Jaffe, D. B., Chang, J. L., Lindblad-Toh, K., Lander, E. S., Koriabine, M., Nefedov, M., Osoegawa, K., Yoshinaga, Y., Zhu, B., and de Jong, P. J. (2007). Identification and analysis of functional elements in 1% of the human genome by the encode pilot project. *Nature*, 447(7146):799–816.

[Feng and Doolittle, 1987] Feng, D. F. and Doolittle, R. F. (1987). Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J Mol Evol*, 25(4):351–60.

[Ferragina and Manzini, 2000] Ferragina, P. and Manzini, G. (2000). Opportunistic data structures with applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 390–398.

[Fousse et al., 2007] Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., and Zimmermann, P. (2007). MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw.*, 33(2):13.

[Gardner et al., 2005] Gardner, P. P., Wilm, A., and Washietl, S. (2005). A benchmark of multiple sequence alignment programs upon structural rnas. *Nucleic Acids Res*, 33(8):2433–9.

[Gautheret and Lambert, 2001] Gautheret, D. and Lambert, A. (2001). Direct rna motif definition and identification from multiple sequence alignments using secondary structure profiles 1 1edited by j. doudna. *Journal of Molecular Biology*, 313(5):1003–1011.

[Gerards, 1995] Gerards, A. (1995). Chapter 3 matching. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 135–224. Elsevier.

[Gorodkin et al., 1997] Gorodkin, J., Heyer, L. J., and Stormo, G. D. (1997). Finding common sequence and structure motifs in a set of rna sequences. *Proc Int Conf Intell Syst Mol Biol*, 5:120–3.

[Gotoh, 1990] Gotoh, O. (1990). Consistency of optimal sequence alignments. *Bull Math Biol*, 52(4):509–25.

[Griffiths-Jones, 2007] Griffiths-Jones, S. (2007). Annotating noncoding RNA genes. *Annu Rev Genomics Hum Genet*, 8:279–98.

[Griffiths-Jones et al., 2003] Griffiths-Jones, S., Bateman, A., Marshall, M., Khanna, A., and Eddy, S. R. (2003). Rfam: an RNA family database. *Nucleic Acids Res*, 31(1):439–41.

[Grillo et al., 2003] Grillo, G., Licciulli, F., Liuni, S., Sbisà, E., and Pesole, G. (2003). Patsearch: A program for the detection of patterns and structural motifs in nucleotide sequences. *Nucleic Acids Res*, 31(13):3608–12.

[Gutell et al., 1992] Gutell, R. R., Power, A., Hertz, G. Z., Putz, E. J., and Stormo, G. D. (1992). Identifying constraints on the higher-order structure of rna: continued development and application of comparative sequence analysis methods. *Nucleic Acids Res*, 20(21):5785–95.

[Haslinger and Stadler, 1999] Haslinger, C. and Stadler, P. F. (1999). Rna structures with pseudo-knots: graph-theoretical, combinatorial, and statistical properties. *Bull Math Biol*, 61(3):437–67.

[Höchsmann, 2005] Höchsmann, M. (2005). *The tree alignment model: algorithms, implementations and applications for the analysis of RNA secondary structures.* PhD thesis, Universität Bielefeld.

[Hofacker et al., 2004] Hofacker, I. L., Bernhart, S. H. F., and Stadler, P. F. (2004). Alignment of rna base pairing probability matrices. *Bioinformatics*, 20(14):2222–7.

[Hofacker et al., 2002] Hofacker, I. L., Fekete, M., and Stadler, P. F. (2002). Secondary structure prediction for aligned RNA sequences. *J Mol Biol*, 319(5):1059–66.

[Hofacker et al., 1994] Hofacker, I. L., Fontana, W., Stadler, P. F., Bonhoeffer, L. S., Tacker, M., and Schuster, P. (1994). Fast folding and comparison of rna secondary structures. *Monatshefte für Chemie/Chemical Monthly*, 125(2):167–188.

[Jabbari et al., 2018] Jabbari, H., Wark, I., Montemagno, C., and Will, S. (2018). Knotty: efficient and accurate prediction of complex rna pseudoknot structures. *Bioinformatics*, 34(22):3849–3856.

[Janssen and Giegerich, 2015] Janssen, S. and Giegerich, R. (2015). The rna shapes studio. *Bioinformatics*, 31(3):423–5.

[Kalvari et al., 2018] Kalvari, I., Nawrocki, E. P., Argasinska, J., Quinones-Olvera, N., Finn, R. D., Bateman, A., and Petrov, A. I. (2018). Non-coding rna analysis using the rfam database. *Current Protocols in Bioinformatics*, 62(1):e51.

[Katoh et al., 2002] Katoh, K., Misawa, K., Kuma, K.-i., and Miyata, T. (2002). Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform. *Nucleic Acids Res*, 30(14):3059–66.

[Katoh and Toh, 2008] Katoh, K. and Toh, H. (2008). Improved accuracy of multiple ncrna alignment by incorporating structural information into a mafft-based framework. *BMC Bioinformatics*, 9:212.

[Kiryu et al., 2007] Kiryu, H., Tabei, Y., Kin, T., and Asai, K. (2007). Murlet: a practical multiple alignment tool for structural rna sequences. *Bioinformatics*, 23(13):1588–98.

[Klein and Eddy, 2003] Klein, R. J. and Eddy, S. R. (2003). Rsearch: finding homologs of single structured rna sequences. *BMC Bioinformatics*, 4:44.

[Knudsen and Hein, 2003] Knudsen, B. and Hein, J. (2003). Pfold: Rna secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Res*, 31(13):3423–8.

[Kozak, 1983] Kozak, M. (1983). Comparison of initiation of protein synthesis in procaryotes, eucaryotes, and organelles. *Microbiol Rev*, 47(1):1–45.

[Laganà et al., 2015] Laganà, A., Veneziano, D., Russo, F., Pulvirenti, A., Giugno, R., Croce, C. M., and Ferro, A. (2015). Computational design of artificial rna molecules for gene regulation. *Methods Mol Biol*, 1269:393–412.

[Lai et al., 2012] Lai, D., Proctor, J. R., Zhu, J. Y. A., and Meyer, I. M. (2012). R-chie: a web server and r package for visualizing rna secondary structures. *Nucleic Acids Res*, 40(12):e95.

[Lai et al., 2003] Lai, E. C., Tomancak, P., Williams, R. W., and Rubin, G. M. (2003). Computational identification of drosophila microRNA genes. *Genome Biol*, 4(7):R42.

[Lalwani et al., 2014] Lalwani, S., Kumar, R., and Gupta, N. (2014). Sequence-structure alignment techniques for rna: a comprehensive survey. *Advances in Life Sciences*, 4(1):21–35.

[Lam et al., 2009] Lam, T. W., Li, R., Tam, A., Wong, S. C. K., Wu, E., and Yiu, S. (2009). High throughput short read alignment via bi-directional BWT. In *2009 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2009,*

Washington, DC, USA, November 1-4, 2009, Proceedings, pages 31–36. IEEE Computer Society.

[Lambert et al., 2004] Lambert, A., Fontaine, J.-F., Legendre, M., Leclerc, F., Permal, E., Major, F., Putzer, H., Delfour, O., Michot, B., and Gautheret, D. (2004). The erpin server: an interface to profile-based rna motif identification. *Nucleic Acids Res*, 32(Web Server issue):W160–5.

[Laslett and Canback, 2004] Laslett, D. and Canback, B. (2004). ARAGORN, a program to detect tRNA genes and tmRNA genes in nucleotide sequences. *Nucleic Acids Res*, 32(1):11–6.

[Liao and Wang, 2004] Liao, B. and Wang, T.-M. (2004). A 3d graphical representation of rna secondary structures. *J Biomol Struct Dyn*, 21(6):827–32.

[Lim and Brown, 2017] Lim, C. S. and Brown, C. M. (2017). Know your enemy: Successful bioinformatic approaches to predict functional rna structures in viral rnas. *Front Microbiol*, 8:2582.

[Lim et al., 2003] Lim, L. P., Lau, N. C., Weinstein, E. G., Abdelhakim, A., Yekta, S., Rhoades, M. W., Burge, C. B., and Bartel, D. P. (2003). The microrna of caenorhabditis elegans. *Genes Dev*, 17(8):991–1008.

[Liu and Wang, 2006] Liu, N. and Wang, T. (2006). A method for rapid similarity analysis of rna secondary structures. *BMC Bioinformatics*, 7(1):493.

[Lorenz et al., 2011] Lorenz, R., Bernhart, S. H., Höner Zu Siederdissen, C., Tafer, H., Flamm, C., Stadler, P. F., and Hofacker, I. L. (2011). Viennarna package 2.0. *Algorithms Mol Biol*, 6:26.

[Losko et al., 2016] Losko, M., Kotlinowski, J., and Jura, J. (2016). Long noncoding rnas in metabolic syndrome related disorders. *Mediators Inflamm*, 2016:5365209.

[Löwes et al., 2016] Löwes, B., Chauve, C., Ponty, Y., and Giegerich, R. (2016). The bralibase dent—a tale of benchmark design and interpretation. *Briefings in Bioinformatics*, 18(2):306–311.

[Lyngsø et al., 1999] Lyngsø, R. B., Zuker, M., and Pedersen, C. N. (1999). Fast evaluation of internal loops in rna secondary structure prediction. *Bioinformatics*, 15(6):440–5.

[Macke et al., 2001] Macke, T. J., Ecker, D. J., Gutell, R. R., Gautheret, D., Case, D. A., and Sampath, R. (2001). RNAMotif, an RNA secondary structure definition and search algorithm. *Nucleic Acids Research*, 29(22):4724–4735.

[Marz et al., 2014] Marz, M., Beerenwinkel, N., Drosten, C., Fricke, M., Frishman, D., Hofacker, I. L., Hoffmann, D., Middendorf, M., Rattei, T., Stadler, P. F., and Töpfer, A. (2014). Challenges in rna virus bioinformatics. *Bioinformatics*, 30(13):1793–9.

[Mathews and Turner, 2002] Mathews, D. H. and Turner, D. H. (2002). Dynalign: an algorithm for finding the secondary structure common to two rna sequences. *J Mol Biol*, 317(2):191–203.

[Matthews, 1975] Matthews, B. W. (1975). Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochim Biophys Acta*, 405(2):442–51.

[Mattick, 2005] Mattick, J. S. (2005). The functional genomics of noncoding rna. *Science*, 309(5740):1527–8.

[McCaskill, 1990] McCaskill, J. S. (1990). The equilibrium partition function and base pair binding probabilities for rna secondary structure. *Biopolymers*, 29(6-7):1105–19.

[Mehlhorn et al., 1996] Mehlhorn, K., Näher, S., and Uhrig, C. (1996). The LEDA platform for combinatorial and geometric computing. In Mayr, H. C., editor, *Beherrschung von Informationssystemen, Tagungsband der Informatik '96, Klagenfurt, Austria, 25.-27. September 1996*, volume 88 of *books@ocg.at*, pages 43–50. Austrian Computer Society.

[Meyer et al., 2011] Meyer, F., Kurtz, S., Backofen, R., Will, S., and Beckstette, M. (2011). Structator: fast index-based search for rna sequence-structure patterns. *BMC Bioinformatics*, 12:214.

[Meyer et al., 2013] Meyer, F., Kurtz, S., and Beckstette, M. (2013). Fast online and index-based algorithms for approximate search of rna sequence-structure patterns. *BMC Bioinformatics*, 14:226.

[Meyer and Miklós, 2007] Meyer, I. M. and Miklós, I. (2007). SimulFold: simultaneously inferring RNA structures including pseudoknots, alignments, and trees using a Bayesian MCMC framework. *PLoS Comput Biol*, 3(8):e149.

[Möhl et al., 2010] Möhl, M., Will, S., and Backofen, R. (2010). Lifting prediction to alignment of rna pseudoknots. *J Comput Biol*, 17(3):429–42.

[Monga and Banerjee, 2019] Monga, I. and Banerjee, I. (2019). Computational identification of pirnas using features based on rna sequence, structure, thermodynamic and physicochemical properties. *Curr Genomics*, 20(7):508–518.

[Nawrocki and Eddy, 2013] Nawrocki, E. P. and Eddy, S. R. (2013). Infernal 1.1: 100-fold faster rna homology searches. *Bioinformatics*, 29(22):2933–2935.

[Nawrocki et al., 2009] Nawrocki, E. P., Kolbe, D. L., and Eddy, S. R. (2009). Infernal 1.0: inference of rna alignments. *Bioinformatics*, 25(10):1335–7.

[Needleman and Wunsch, 1970] Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48(3):443–53.

[Notredame et al., 2000] Notredame, C., Higgins, D. G., and Heringa, J. (2000). T-coffee: a novel method for fast and accurate multiple sequence alignment 1 1edited by j. thornton. *Journal of Molecular Biology*, 302(1):205–217.

[Nussinov and Jacobson, 1980] Nussinov, R. and Jacobson, A. B. (1980). Fast algorithm for predicting the secondary structure of single-stranded rna. *Proc Natl Acad Sci U S A*, 77(11):6309–13.

[Olson et al., 2009] Olson, W. K., Esguerra, M., Xin, Y., and Lu, X.-J. (2009). New information content in rna base pairing deduced from quantitative analysis of high-resolution structures. *Methods*, 47(3):177–86.

[Palazzo and Lee, 2015] Palazzo, A. F. and Lee, E. S. (2015). Non-coding rna: what is functional and what is junk? *Front Genet*, 6:2.

[Pałkowski and Bielecki, 2019] Pałkowski, M. and Bielecki, W. (2019). Parallel cache-efficient code for computing the McCaskill partition functions. In Ganzha, M., Maciaszek, L. A., and Paprzycki, M., editors, *Proceedings of the 2019 Federated Conference on Computer Science and Information Systems, FedCSIS 2019, Leipzig, Germany, September 1-4, 2019*, volume 18 of *Annals of Computer Science and Information Systems*, pages 207–210.

[Parker et al., 2016] Parker, N., Schneegurt, M., Tu, A.-H. T., Lister, P., and Forster, B. M. (2016). *Microbiology*, chapter 10.3. OpenStax, Houston, Texas.

[Pervouchine, 2018] Pervouchine, D. D. (2018). Towards long-range rna structure prediction in eukaryotic genes. *Genes (Basel)*, 9(6).

[Picardi, 2015] Picardi, E., editor (2015). *RNA Bioinformatics*. Number 1269 in Methods in Molecular Biology. Humana Press, New York.

[Pockrandt et al., 2017] Pockrandt, C., Ehrhardt, M., and Reinert, K. (2017). Eprdictionaries: A practical and fast data structure for constant time searches in unidirectional and bidirectional FM indices. In Sahinalp, S. C., editor, *Research in Computational Molecular Biology - 21st Annual International Conference, RECOMB 2017, Hong Kong, China, May 3-7, 2017, Proceedings*, volume 10229 of *Lecture Notes in Computer Science*, pages 190–206.

[Pockrandt, 2015] Pockrandt, C. M. (2015). Generic implementation of a bidirectional FM-index in SeqAn and applications. Master's thesis, Freie Universität Berlin.

[Ponting et al., 2009] Ponting, C. P., Oliver, P. L., and Reik, W. (2009). Evolution and functions of long noncoding rnas. *Cell*, 136(4):629–41.

[Prince et al., 2022] Prince, S., Munoz, C., Filion-Bienvenue, F., Rioux, P., Sarrasin, M., and Lang, B. F. (2022). Refining mitochondrial intron classification with erpin:

Identification based on conservation of sequence plus secondary structure motifs. *Front Microbiol*, 13:866187.

[Qian et al., 2019] Qian, X., Zhao, J., Yeung, P. Y., Zhang, Q. C., and Kwok, C. K. (2019). Revealing lncrna structures and interactions by sequencing-based approaches. *Trends Biochem Sci*, 44(1):33–52.

[Quast et al., 2013] Quast, C., Pruesse, E., Yilmaz, P., Gerken, J., Schweer, T., Yarza, P., Peplies, J., and Glöckner, F. O. (2013). The silva ribosomal rna gene database project: improved data processing and web-based tools. *Nucleic Acids Res*, 41(Database issue):D590–6.

[Rahn et al., 2018] Rahn, R., Budach, S., Costanza, P., Ehrhardt, M., Hancox, J., and Reinert, K. (2018). Generic accelerated sequence alignment in seqan using vectorization and multi-threading. *Bioinformatics*, 34(20):3437–3445.

[Rausch et al., 2008] Rausch, T., Emde, A.-K., Weese, D., Doring, A., Notredame, C., and Reinert, K. (2008). Segment-based multiple sequence alignment. *Bioinformatics*, 24(16):i187–i192.

[Reinert et al., 2017] Reinert, K., Dadi, T. H., Ehrhardt, M., Hauswedell, H., Mehringer, S., Rahn, R., Kim, J., Pockrandt, C., Winkler, J., Siragusa, E., Urgese, G., and Weese, D. (2017). The SeqAn C++ template library for efficient sequence analysis: A resource for programmers. *J Biotechnol*, 261:157–168.

[Resta, 2018] Resta, R. (2018). Development of a graph reduction method for minimizing RNA secondary structures and speeding-up sequence-structure alignment algorithms. Master's thesis, Politecnico di Torino.

[Reuter and Mathews, 2010] Reuter, J. S. and Mathews, D. H. (2010). Rnastructure: software for rna secondary structure prediction and analysis. *BMC Bioinformatics*, 11:129.

[Rivas et al., 2017] Rivas, E., Clements, J., and Eddy, S. R. (2017). A statistical test for conserved rna structure shows lack of evidence for structure in lncrnas. *Nat Methods*, 14(1):45–48.

[Rivas and Eddy, 1999] Rivas, E. and Eddy, S. R. (1999). A dynamic programming algorithm for rna structure prediction including pseudoknots. *J Mol Biol*, 285(5):2053–68.

[Roehr et al., 2017] Roehr, J. T., Dieterich, C., and Reinert, K. (2017). Flexbar 3.0 - simd and multicore parallelization. *Bioinformatics*, 33(18):2941–2942.

[Sankoff, 1985] Sankoff, D. (1985). Simultaneous solution of the rna folding, alignment and protosequence problems. *SIAM journal on applied mathematics*, 45(5):810–825.

[Sato et al., 2011] Sato, K., Kato, Y., Hamada, M., Akutsu, T., and Asai, K. (2011). Ipknot: fast and accurate prediction of rna secondary structures with pseudoknots using integer programming. *Bioinformatics*, 27(13):i85–93.

[Schnattinger et al., 2012] Schnattinger, T., Ohlebusch, E., and Gog, S. (2012). Bidirectional search in a string with wavelet trees and bidirectional matching statistics. *Inf. Comput.*, 213:13–22.

[Seemann et al., 2011] Seemann, S. E., Menzel, P., Backofen, R., and Gorodkin, J. (2011). The PETfold and PETcofold web servers for intra- and intermolecular structures of multiple RNA sequences. *Nucleic Acids Res*, 39(Web Server issue):W107–11.

[Shapiro and Zhang, 1990] Shapiro, B. A. and Zhang, K. Z. (1990). Comparing multiple rna secondary structures using tree comparisons. *Comput Appl Biosci*, 6(4):309–18.

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

[Sorescu et al., 2012] Sorescu, D. A., Möhl, M., Mann, M., Backofen, R., and Will, S. (2012). CARNA–alignment of RNA structure ensembles. *Nucleic Acids Res*, 40(Web Server issue):W49–53.

[Spitale et al., 2014] Spitale, R. C., Flynn, R. A., Torre, E. A., Kool, E. T., and Chang, H. Y. (2014). Rna structural analysis by evolving shape chemistry. *Wiley Interdiscip Rev RNA*, 5(6):867–81.

[Staple and Butcher, 2005] Staple, D. W. and Butcher, S. E. (2005). Pseudoknots: Rna structures with diverse functions. *PLoS Biol*, 3(6):e213.

[Strauch, 2017] Strauch, B. (2017). Fuzzy RNA motifs for index-based RNA family search. Master's thesis, Freie Universität Berlin.

[Sun and Kraus, 2015] Sun, M. and Kraus, W. L. (2015). From discovery to function: the expanding roles of long noncoding rnas in physiology and disease. *Endocr Rev*, 36(1):25–64.

[Sun et al., 2012] Sun, Y., Buhler, J., and Yuan, C. (2012). Designing filters for fast-known ncrna identification. *IEEE/ACM Trans Comput Biol Bioinform*, 9(3):774–87.

[Szymanski et al., 2002] Szymanski, M., Barciszewska, M. Z., Erdmann, V. A., and Barciszewski, J. (2002). 5s ribosomal rna database. *Nucleic Acids Res*, 30(1):176–8.

[Tabei et al., 2008] Tabei, Y., Kiryu, H., Kin, T., and Asai, K. (2008). A fast structural multiple alignment method for long rna sequences. *BMC Bioinformatics*, 9(1):33.

[Tabei et al., 2006] Tabei, Y., Tsuda, K., Kin, T., and Asai, K. (2006). Scarna: fast and accurate structural alignment of rna sequences by matching fixed-length stem fragments. *Bioinformatics*, 22(14):1723–9.

[Tan et al., 2017] Tan, Z., Fu, Y., Sharma, G., and Mathews, D. H. (2017). Turbofold ii: Rna structural alignment and secondary structure prediction informed by multiple homologs. *Nucleic Acids Res*, 45(20):11570–11581.

[Thompson et al., 1994] Thompson, J. D., Higgins, D. G., and Gibson, T. J. (1994). Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22(22):4673–80.

[Torarinsson et al., 2007] Torarinsson, E., Havgaard, J. H., and Gorodkin, J. (2007). Multiple structural alignment and clustering of rna sequences. *Bioinformatics*, 23(8):926–932.

[Viehweger et al., 2019] Viehweger, A., Krautwurst, S., Lamkiewicz, K., Madhugiri, R., Ziebuhr, J., Hölzer, M., and Marz, M. (2019). Direct rna nanopore sequencing of full-length coronavirus genomes provides novel insights into structural variants and enables modification analysis. *Genome Res*, 29(9):1545–1554.

[Vingron and Argos, 1990] Vingron, M. and Argos, P. (1990). Determination of reliable regions in protein sequence alignments. *Protein Eng*, 3(7):565–9.

[Warnow, 2021] Warnow, T. (2021). Revisiting evaluation of multiple sequence alignment methods. *Methods Mol Biol*, 2231:299–317.

[Wei et al., 2011] Wei, D., Alpert, L. V., and Lawrence, C. E. (2011). Rnag: a new gibbs sampler for predicting rna secondary structure for unaligned sequences. *Bioinformatics*, 27(18):2486–93.

[Wheeler and Eddy, 2013] Wheeler, T. J. and Eddy, S. R. (2013). nhmmer: DNA homology search with profile HMMs. *Bioinform.*, 29(19):2487–2489.

[Will et al., 2012] Will, S., Joshi, T., Hofacker, I. L., Stadler, P. F., and Backofen, R. (2012). Locarna-p: Accurate boundary prediction and improved detection of structural rnas. *RNA*, 18(5):900–914.

[Will et al., 2015] Will, S., Otto, C., Miladi, M., Möhl, M., and Backofen, R. (2015). Sparse: quadratic time simultaneous alignment and folding of rnas without sequence-based heuristics. *Bioinformatics*, 31(15):2489–96.

[Will et al., 2007] Will, S., Reiche, K., Hofacker, I. L., Stadler, P. F., and Backofen, R. (2007). Inferring noncoding rna families and classes by means of genome-scale structure-based clustering. *PLoS Comput Biol*, 3(4):e65.

[Will et al., 2013] Will, S., Yu, M., and Berger, B. (2013). Structure-based whole-genome realignment reveals many novel noncoding rnas. *Genome Res*, 23(6):1018–27.

[Wilm et al., 2008] Wilm, A., Higgins, D. G., and Notredame, C. (2008). R-coffee: a method for multiple alignment of non-coding rna. *Nucleic Acids Res*, 36(9):e52.

[Wilusz et al., 2009] Wilusz, J. E., Sunwoo, H., and Spector, D. L. (2009). Long noncoding rnas: functional surprises from the rna world. *Genes Dev*, 23(13):1494–504.

[Winkler et al., 2022] Winkler, J., Urgese, G., Ficarra, E., and Reinert, K. (2022). LaRA 2: parallel and vectorized program for sequence-structure alignment of RNA sequences. *BMC Bioinformatics*, 23(1):18.

[Wolf et al., 2005] Wolf, M., Achtziger, M., Schultz, J., Dandekar, T., and Müller, T. (2005). Homology modeling revealed more than 20,000 rrna internal transcribed spacer 2 (its2) secondary structures. *RNA*, 11(11):1616–23.

[Wuchty et al., 1999] Wuchty, S., Fontana, W., Hofacker, I. L., and Schuster, P. (1999). Complete suboptimal folding of rna and the stability of secondary structures. *Biopolymers*, 49(2):145–65.

[Xu and Mathews, 2011] Xu, Z. and Mathews, D. H. (2011). Multilign: an algorithm to predict secondary structures conserved in multiple rna sequences. *Bioinformatics*, 27(5):626–32.

[Yao et al., 2019] Yao, R.-W., Wang, Y., and Chen, L.-L. (2019). Cellular functions of long noncoding RNAs. *Nat Cell Biol*, 21(5):542–551.

[Yasnev, 2015] Yasnev, O. (2015). Adaptation of multiple sequence alignment algorithm from the SeqAn library for processing deep alignments. Master's thesis, Saint Petersburg Academic University.

[Zadeh et al., 2011] Zadeh, J. N., Steenberg, C. D., Bois, J. S., Wolfe, B. R., Pierce, M. B., Khan, A. R., Dirks, R. M., and Pierce, N. A. (2011). Nupack: Analysis and design of nucleic acid systems. *J Comput Chem*, 32(1):170–3.

[Zhao and Sahni, 2020] Zhao, C. and Sahni, S. (2020). Efficient computation of RNA partition functions using McCaskill's algorithm. In *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*, pages 449–452.

[Zuker and Stiegler, 1981] Zuker, M. and Stiegler, P. (1981). Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Res*, 9(1):133–48.

Glossary

API Application programming interface, a documented set of functions of a software that is exposed for usage by other software. 27, 31, 32, 47, 76, 93

ATP Adenosine triphosphate is an organic compound that transfers energy in living cells to drive various processes. 7

index The term index has a lot of meanings in different contexts – in this thesis we use two of them: (**1**) An index is a non-negative integral number that denotes the absolute position of an element within a sequence, string, array or matrix. It is often used as a key or identifier of a specific element. Unless noted otherwise the first element has index 0, see *index sequence*. (**2**) An index data structure allows fast searches in an underlying text. In part III of this thesis we use a bi-directional FM-index for searching patterns in a genome. 22, 29, 31, 33, 38, 55, 85, 94, 98, 108, 115

index sequence An integer sequence $(0, 1, 2, \ldots, L-1)$ of length L. Each value represents its own positional *index* (here 0-based). 38, 39, 48

matching A matching in an undirected graph is a set of edges without common vertices. In a weighted graph the maximum-weighted matching is a matching in which the sum of weights is maximal. 21, 36, 39, 42, 43, 49, 50, 69, 115

MSA Multiple sequence alignment; opposed to a pairwise alignment, this is an alignment of more than two sequences. 17, 53, 56, 65

mutex A mutual exclusion is used to control the access to shared resources. It ensures that only one thread can enter the critical section at a time. 49, 96

NMR spectroscopy Nuclear magnetic resonance spectroscopy is a method to detect magnetic fields around atoms. These magnetic fields reveal details of the electronic structure of a molecule and its individual functional groups. 16

primary structure The specification of the atomic composition of a biopolymer, for RNA/DNA it is equivalent to the sequence. 3

pseudoknot Structural interactions between loop regions that cross each other, see figure 1.4b for a visualization. 9, 11, 13, 15, 22, 59, 66, 69, 73, 75, 77, 111, 112, 115

PSSM Position-specific score matrix (also called *position weight matrix*) is a scoring system for sequence alignment that provides a score dependent on the position within the sequence, rather than the sequence character itself. 21, 36, 38, 42, 47, 48

secondary structure The pattern of intramolecular base pairings. 3, 7, 22, 55, 67, 70, 73, 115

sequence Denotes the type and order of nucleotides or amino acids in a biopolymer, in this work usually the nucleotides in an RNA molecule. 3, 5, 21, 73, 115

SHAPE Selective 2' Hydroxyl Acylation and Primer Extension: A type of experiment for RNA that determines how likely a nucleotide is paired or unpaired. 29, 54, 69, 70

SIMD Single Instruction, Multiple Data is a type of parallel processing, where the same operation is performed on multiple data points simultaneously. 45, 48

STAU1 Double-stranded RNA-binding protein Staufen homolog 1 is a protein that in humans is encoded by the STAU1 gene. 6

stdout The standard output is a communication channel that prints the output of a program back to the command-line. 53

STL The Standard Template Library is a powerful set of C++ template classes. It provides general-purpose classes and functions with templates that implement many popular and commonly used algorithms and data structures. 79

transposable element A gene that can change its position in the genome, also called *jumping gene*. 5

WUSS Washington University Secondary Structure notation. A linear representation of secondary structure with pseudoknot support. 9, 75

Zusammenfassung

Die Funktion nicht-kodierender RNA-Sequenzen wird weitgehend durch ihre räumliche Konformation bestimmt. Dies ist die Sekundärstruktur des Moleküls, welche durch Watson-Crick-Wechselwirkungen zwischen Nukleotiden gebildet wird. Daher berücksichtigen moderne RNA-Alignment-Algorithmen routinemäßig Strukturinformationen. Wesentliche Aufgaben um noch unbekannte RNA-Familien zu entdecken und auf ihre möglichen Funktionen zu schließen, sind das strukturelle Alignment von RNAs und die anschließende Suche nach den abgeleiteten Strukturmotiven. Diese Aufgaben erfordern einen hohen Rechenaufwand, insbesondere für das Alignment vieler langer Sequenzen und erfordern daher effiziente Algorithmen, die moderne Hardware (soweit verfügbar) optimal ausnutzen. Einige der Sekundärstrukturen enthalten überlappende Interaktionen, sogenannte Pseudoknoten, welche die Analyse zusätzlich komplexer machen und von bereits bestehender Software oft ignoriert werden.

In dieser Arbeit stelle ich mit LaRA 2 und MaRs zwei SeqAn-basierte Software-Tools vor, die Algorithmen zum Auffinden von Sequenzstrukturmotiven in genomischen Sequenzen implementieren. Im Gegensatz zu anderen Programmen können meine Programme beliebige Pseudoknoten verarbeiten. Sie nutzen Multithreading zur gleichzeitigen Ausführung von Programmteilen und sind in modernem C++-Code implementiert, um maximale Langlebigkeit und Leistung zu gewährleisten.

LaRA 2 ist deutlich schneller als vergleichbare Software für paarweise und multiple Alignments von strukturierten RNA-Sequenzen. Es verwendet eine neue Heuristik zur Berechnung einer unteren Schranke zur Lösung und setzt Vektorisierungstechniken ein, um die zeitkritischen Bestandteile des Algorithmus zu beschleunigen.

MaRs kann in einem Workflow direkt an das Ergebnis von LaRA 2 angesetzt werden und leitet Sequenz-Struktur-Motive aus den strukturellen Alignments ab. Die Motive sind Deskriptoren für die Eigenschaften der RNA-Sequenzen und steuern die Suche nach homologen Sequenzen in einem Genom. MaRs verwendet einen bidirektionalen Index für die Genomsequenzen und eine optimierte, parallel ausführbare Suchstrategie, um die Übereinstimmungen extrem schnell zu finden. Die Verwendung eines Thread-Pools, effektive Pruning-Strategien und ein geringer Speicherbedarf sorgen dafür, dass MaRs in der Lage ist, auch extrem große Datensätze zu verarbeiten.

Declaration of authorship

I declare to the Freie Universität Berlin that I have completed the submitted dissertation independently and without the use of sources and aids other than those indicated. The present thesis is free of plagiarism. I have marked as such all statements that are taken literally or in content from other writings. This dissertation has not been submitted in the same or similar form in any previous doctoral procedure. I agree to have my thesis examined by a plagiarism examination software.

Selbstständigkeitserklärung

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Dissertation selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe. Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht. Diese Dissertation wurde in gleicher oder ähnlicher Form noch in keinem früheren Promotionsverfahren eingereicht. Mit einer Prüfung meiner Arbeit durch ein Plagiatsprüfungsprogramm erkläre ich mich einverstanden.

Berlin, 18.09.2022