Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics
Department of Computer Science

Computational Complexity and Practical Implementation of RNA Motif Search

(Master's Thesis)

2012

Bc. Ladislav Rampášek

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Computational Complexity and Practical Implementation of RNA Motif Search

Master's Thesis

Study Program: Computer Science
Branch of Study: 2508 Informatika
Supervisor: Mgr. Bronislava Brejová PhD.

Bratislava, 2012

Bc. Ladislav Rampášek

15218056

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Ladislav Rampášek
Študijný program:	informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	9.2.1. informatika
Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	anglický

Názov: Výpočtová zložitosť a praktická implementácia hľadania RNA motívov

Cieľ: Na vyhľadávanie RNA génov v genómoch biológovia používajú štrukturálne motívy pripomínajúce regulárne výrazy. Cieľom práce je študovať výpočtovú zložitosť ich vyhľadávania a vyvinúť metódy, ktoré dobre fungujú na praktických príkladoch.

Vedúci: Mgr. Bronislava Brejová, PhD.
Katedra: FMFI.KI - Katedra informatiky

Dátum zadania: 19.10.2010

Dátum schválenia: 27.04.2012 prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

....

študent vedúci práce

15218056

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Ladislav Rampášek
Study programme:	Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	9.2.1. Computer Science, Informatics
Type of Thesis:	Diploma Thesis
Language of Thesis:	English

Title: Computational Complexity and Practical Implementation of RNA Motif Search

Aim: To search for RNA genes in genomes, biologists use structural motifs similar to regular expressions. The goal of the thesis is to study computational complexity of searching for occurrences of such motifs and to develop a practical algorithm for such search.

Supervisor:	Mgr. Bronislava Brejová, PhD.
Department:	FMFI.KI - Department of Computer Science
Assigned:	19.10.2010
Approved:	27.04.2012

prof. RNDr. Branislav Rovan, PhD.
Guarantor of Study Programme

....
Student

....
Supervisor

I hereby declare that I wrote this thesis by myself, only with the help of the referenced literature, under the careful supervision of my thesis advisor.

. . . .

Acknowledgments

I am deeply grateful to my supervisor Dr. Broňa Brejová, and Prof. Andrej Lupták for their invaluable help and guidance, many insightful conversations and comments during the development of my thesis.

I want to especially acknowledge Prof. Andrej Lupták for motivating this research, for invaluable hands-on experience and close guidance in the world of RNA. Research fellowship in Prof. Andrej Lupták's research lab at the University of California, Irvine significantly added to the quality of my work.

Further, I was privileged to collaborate with Dr. Tomáš Vinař, and with Randi M. Jimenez, whose inspiring comments and precious hands-on experience significantly helped in the development of practical software tools.

I would like to also acknowledge the National Scholarship Programme of the Slovak Republic and the Tatrabanka Foundation for their generous financial support.

Ladislav Rampášek

Abstrakt

Autor:	Bc. Ladislav Rampášek
Názov práce:	Výpočtová zložitosť a praktická implementácia hľadania RNA motívov
Univerzita:	Univerzita Komenského v Bratislave
Fakulta:	Fakulta matematiky, fyziky a informatiky
Katedra:	Katedra informatiky
Vedúci diplomovej práce:	Mgr. Bronislava Brejová PhD.
Rozsah práce:	59 strán
Dátum:	máj 2012

V tejto práci sme sa venovali problému vyhľadávania štrukturálnych RNA motívov. Naša práca je motivovaná výskumom Prof. Andreja Luptáka [Ruminski et al., 2010] v oblasti kataliticky aktívnych RNA, tzv. ribozýmov. Táto práca nadväzuje na našu predchádzajúcu prácu [Rampášek, 2010], v ktorej sme navrhli nový algoritmus na vyhľadávanie RNA motívov založený na publikovanom prehľadávaní s návratom [Gautheret et al., 1990]. Náplňou tejto diplomovej práce sú tri hlavné výsledky. Za prvé, dokázali sme NP-úplnosť problému vyhľadávania RNA štrukturálnych motívov pomocou redukcie z problému ONE-IN-THREE 3SAT. Ďalej, navrhli sme adaptívnu metódu pre usporiadanie elementov v prehľadávaní s návratom použitom v RNArobo. Túto metódu sme implementovali v nástroji RNArobo 2.0. Táto zmena priniesla značné zrýchlenie. Pre zložité motívy je tak RNArobo 2.0 rýchlejší ako zaužívané nástroje RNAbob [Eddy, 1996] a RNAMotif [Macke et al., 2001]. Na záver, vyvinuli sme nástroje, ktoré umožňujú ohodnotiť nájdené výskyty daného motívu na základe odhadu stability ich štruktúry.

Celkovo naša práca vyústila v sadu nástrojov, ktoré umožňujú automatizovaný postup využiteľný pre objavovanie nových výskytov funkčných RNA, podobný postupu navrhnutému v [Jimenez et al., 2012].

KĽÚČOVÉ SLOVÁ: RNA motív, vyhľadávanie v texte, NP-úplnosť, prehľadávanie s návratom, RNArobo

Abstract

Author:	Bc. Ladislav Rampášek
Title:	Computational Complexity and Practical Implementation of RNA Motif Search
University:	Comenius University in Bratislava
Faculty:	Faculty of Mathematics, Physics and Informatics
Department:	Departement of Computer Science
Supervisor:	Mgr. Bronislava Brejová PhD.
Number of Pages:	59
Date:	May 2012

In this thesis we study the RNA structural motif search problem. Our work is motivated by the research of Prof. Andrej Lupták in the area of self-cleaving ribozymes [Ruminski et al., 2010]. This thesis follows our previous work [Rampášek, 2010], where we proposed a new algorithm for RNA motif search based on a published backtracking method [Gautheret et al., 1990]. Here, we present three main results. Firstly, we have proven the RNA structural motif search problem to be NP-complete by a reduction from ONE-IN-THREE 3SAT. Secondly, we have devised a data-driven element ordering strategy for the backtracking search of RNArobo. We have implemented this strategy in RNArobo 2.0. This change has considerably improved the running time, and for complex motifs RNArobo 2.0 outperforms other search tools, RNAbob [Eddy, 1996] and RNAMotif [Macke et al., 2001]. Finally, we have developed tools for post-processing RNArobo search results by ranking them according to their estimated structural stability.

Overall our work resulted in a practical computational pipeline, similar to that proposed in [Jimenez et al., 2012], that can be used to discover new homologs of functional RNAs.

KEYWORDS: RNA motif, pattern matching, NP-completeness, backtracking, RNArobo

Contents

Introduction

In this thesis we study the problem of RNA structural motif search, which originates in computational biology. Our work is motivated by the research of Prof. Andrej Lupták in the area of self-cleaving ribozymes [Ruminski et al., 2010], that are functional RNA molecules.

The problem of RNA structural motif search came to foreground with the discovery of many novel functional RNA molecules, starting in the 1980s and continuing to present days. At the same time there is an abundant amount of genomic data available. Since functional RNAs are encoded in the genomic DNA, we can use computational tools to discover new RNAs by DNA sequencing analysis. Functional RNAs can be characterized by patterns resembling regular expressions, enhanced with relational properties that may cause the pattern to be context sensitive.

Many computational methods have been proposed since the early 1990s to address the problem of RNA motif search. Some of the available search tools are strictly specialized for particular types of RNAs, and the general-purpose tools differ in the class of possible RNA structural features considered in the search. In [Rampášek, 2010] we proposed a new algorithm based on previous work of [Gautheret et al., 1990], that allows to search for motifs with single-letter insertions, which was an option missing from existing tools. However running time of our implementation was considerably slower than that of the other tools.

In this thesis we follow our previous work, with the following two main goals. Firstly, we want to study the computational complexity of the RNA motif search problem. Secondly, we want to enhance our previous algorithm and its implementation to enable genome-wide searches in a reasonable time. For example, the human genome is approximately 3×10^9 letters long, and we would like to be able to search through it in several hours.

We begin by introducing the biology background of the problem. Later in the first chapter we also give a brief overview of the existing RNA motif search tools.

We devote Chapter 2 to the study of the computational complexity of the RNA motif search, which we prove to be NP-complete. We formally define the problem and prove its NP-completeness by a reduction from ONE-IN-THREE 3SAT. We conclude this chapter by a discussion on the causes of the NP-completeness.

In Chapter 3 we summarize our previously proposed algorithm from [Rampášek, 2010], called RNArobo. RNArobo is based on a backtracking search [Gautheret et al., 1990] with dynamic programming for finding occurrences of individual elements of the motif.

In Chapter 4 we propose a data-driven element ordering strategy for the backtracking search of RNArobo. This method is based on heuristic choice of good candidate orders and their further empirical evaluation and statistical elimination.

We devote Chapter 5 to description of a method for RNArobo search results post-processing. We automatized an *in silico* pipeline used by Prof. Andrej Lupták to test a functionality of found motif occurrences, as it is intractable to test all found RNA sequences *in vitro*.

We conclude this thesis by Chapter 6 dedicated to experimental results of our work. We measure execution time of our enhanced tool RNArobo 2.0 on real biological data, and compare it to several established tools.

Chapter 1

Biological Background

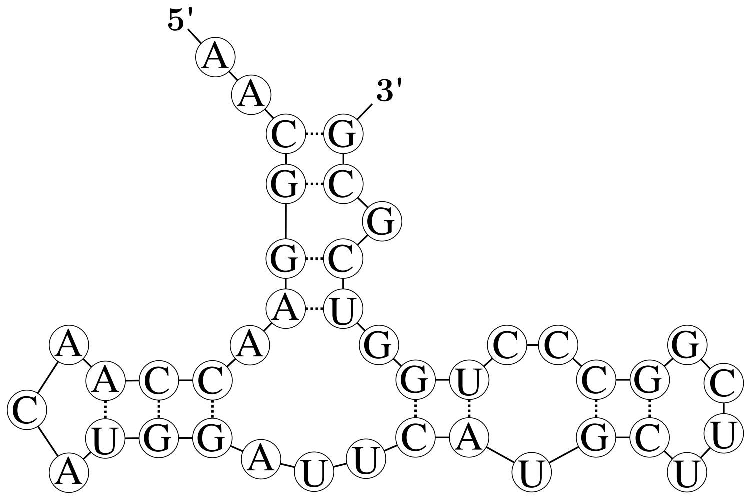


Figure 1.1: An example of an RNA structure. Full lines represent the order of the sequence from 5' beginning to 3' end. Dotted lines represent base pairs that form the secondary structure.

Ribonucleic acid (RNA) is a polymer constructed from nucleotides, similar to DNA. A chain of nucleotides is called an RNA sequence or an RNA primary structure, whose ends are marked 5' and 3', respectively. The four nucleotides are abbreviated `A`, `C`, `G` and `U`, which stand for Adenine, Cytosine, Guanine and Uracil, respectively. We will refer to individual nucleotides also as bases or residues. In contrast, DNA uses thymine (`T`) instead of uracil. The nucleotide sequences of RNAs produced by an organism are generally encoded in the genomic DNA of the organism and then, by the process called transcription, copied to the form of RNA. Therefore scientists search for RNA motifs directly in DNA. In this thesis we will threat nucleotides `U` and `T` as equal and will not distinguish between them.

RNA does not form a double helix like DNA does, instead it is usually single stranded, but may have complex three-dimensional structure due to complementary bonds between the parts of the same strand. This happens as some nucleotides can bind together to form a pair. In particular `G`-`C` and `A`-`U` (`A`-`T` in DNA) form hydrogen bonded base pairs and are said to be complementary. These two canonical base pairs are also known as Watson-Crick pairs, named after James D. Watson and Francis Crick, who discovered the helical DNA structure in 1953. In addition to canonical base pairs, non-canonical pairs also occur in RNA secondary structure. The most common non-canonical pair is G-U pair, which is almost as thermodynamically stable as Watson-Crick pairs. The base-paired structure is called the secondary structure of the RNA, and the form which it adopts in 3D space is called the tertiary structure. An example of an RNA secondary structure can be seen in Figure 1.1. For many purposes, the secondary structure is a good approximation of the tertiary structure as it contains enough information on the spatial structure, while keeping the complexity and time performance of related algorithms at reasonable level.

The first widely known RNA function was its role of a passive intermediary messenger in the process of translation of proteins from DNA. In this scenario, RNA encodes information about the composition of a protein. Therefore this type of RNA is called coding RNA.

However, subsequent research has shown that there exist many RNAs that do not encode any protein. These types of RNA are called non-coding RNAs (ncRNA). Non-coding RNAs adopt sophisticated three-dimensional structures. They are involved in gene regulation, RNA processing, and other roles. For this type of RNA, the structure to which it folds in space is often more important that the actual sequence.

New technologies allow us to obtain DNA sequences of various species. Given known examples of functional RNAs, we would like to locate RNAs with similar function (homologs) in newly sequenced genomes. One option is to use searches based purely on sequence similarity. However, often we obtain higher sensitivity by allowing sequence to vary and searching for certain typical structural motifs [Webb et al., 2009]. RNA structural motifs generally define relative positions of base pairs in a secondary structure. Our goal is then to search for regions in DNA that could form the secondary structure prescribed by the motif.

Base pairs in the RNA secondary structure typically form a number of short base-paired stems (or also called as helices). Helices may overlap to form so-called pseudoknots. This happens when a helix starts before another one has already ended, this irregularity is similar to an incorrectly parenthesized expression, e.g. "([ )]". An example of a pseudoknotted structure is shown in Figure 1.2.

Pseudoknots make some computational problems harder, often being NP-complete [Durbin et al., 1998, Lyngsø, 2004]. Therefore for many purposes, as for example database searching for RNA homologs or RNA secondary structure prediction, it is usually acceptable to sacrifice the information in pseudoknots in return for efficient algorithms [Durbin et al., 1998]. In this thesis we consider pseudoknots without any restrictions and thus allow arbitrarily pseudoknotted structures.

More formally, we can define a secondary structure of an RNA strand of length n indexed $1 \ldots n$ (starting at the 5' end) as a set of pairing interactions (i, j) between i-th and j-th base of the strand, where every base may participate in at most one pair. By this definition, we can distinguish two fundamental elements of RNA secondary structure – single-stranded regions and helical (paired) regions.

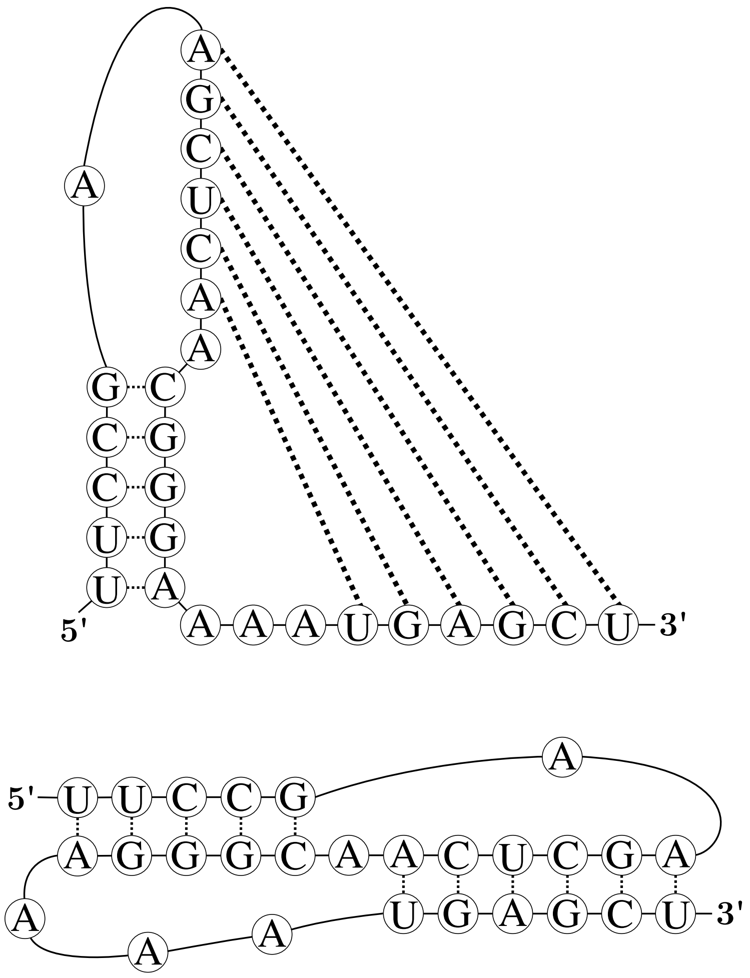

5'
3'
U U C C G A G C C A A A G G G C A A A A U G A G C U
5'
U U C C G A A G G G C A A C U C G A A A A U G A G C U
3'

Figure 1.2: Two different representations of a pseudoknotted structure. The representation in the top emphasizes which base interactions (thick dotted lines) cause the pseudoknot.

1.1 Existing Search Tools

The problem of RNA structural motif search was first addressed in late 1980s. The first tool enabling the structural search was RNAMot [Gautheret et al., 1990], which introduced a backtracking method. RNAbob [Eddy, 1996] is its more efficient reimplementation based on a nondeterministic finite state automaton with node rewriting rules. Afterwards, numerous stand-alone and web-service based tools were developed allowing users to define RNA motifs according to their requirements and search for these motifs in sequence databases. These tools differ in the way a user defines a structure that is to be searched for, then in allowing and handling defects permitted in found matches (e.g. mismatches, mispairing, insertions), and post-processing filters applied on found matches. Examples include Palingol [Billoud et al., 1996], RNAMotif [Macke et al., 2001], PatSearch [Grillo et al., 2003], RNAMST [Chang et al., 2006], Locomotif [Reeder et al., 2007]; for an extensive overview of current search tools see [George and Tenenbaum, 2009]. Besides general-purpose motif search tools, tools optimized for search of specific structural RNA molecules were devised as well, e.g. tRNAscan-SE [Lowe and Eddy, 1997].

In 2010 we implemented a new search tool RNArobo [Rampášek, 2010], which builds on the motif format developed for RNAMot and RNAbob, but in addition enables a user to permit insertions to the structure. Algorithms searching for motifs including pseudoknots in general have exponential time complexity. In 2011 we proved the RNA secondary structure motif search problem to be NP-complete [Rampášek, 2011]. This proof is the subject of the next chapter.

Chapter 2

Computational Complexity

In this section we formally define the problem in a form suitable for the analysis of computational complexity. It is important to note here, that search tools usually solve a more general problem. Our simpler definition captures the essence of the problem and suffices to show its NP-hardness. First, we will define a structural motif – an abstraction of RNA structural motif.

2.1 Structural Motif

Structural motif specifies both primary and secondary structure constraints. The primary structure deals with sequence constraints, while the secondary structure describes the relations of the nucleotide bases forming the primary structure. By the definition of secondary structure as a set of base pairings, we can distinguish two fundamental elements of RNA structure – *single-stranded regions* and *helical (paired) regions.* On this observation we are going to built the definition of a structural motif, as an abstraction of RNA structural motif. First we introduce necessary terminology.

Sequence $A = a_1 \dots a_n$ is a string over a finite alphabet Σ.

Sequence constraint alphabet Σ_+ will be used to express constraints on primary structure and we define it as

$$\Sigma_+ = \left(\mathcal{P}\left(\Sigma\right) \smallsetminus \{\emptyset\}\right) \cup \left\{\{*\}\right\}$$

where '$*$' is a wildcard which matches every character from Σ or may be omitted, so that patterns of variable length are possible. Note that Σ_+ is not a set of characters, but a set of sets of characters.

Fitting function FIT describes which symbols satisfy a given sequence constraint:

$$FIT : \Sigma_+ \times \Sigma \to \{0, 1\}$$

$$\forall s \in \Sigma_+, \forall a \in \Sigma$$

$$FIT(s, a) = \begin{cases} 1 & \textit{iff } \; a \in s \vee s = \{*\} \\ 0 & \textit{otherwise} \end{cases}$$

We will state later on, how a wildcard '$*$' may be left out.

Single-stranded element $S = s_1 \dots s_n$ is a string over the alphabet Σ_+.

Helical element (H, H') represents a paired region, thus consists of two parts H and H'. Both $H = s_1 \dots s_m$ and $H' = s'_1 \dots s'_m$ are strings of the same length $m \in \mathbb{N}$ over the alphabet Σ_+.

In a valid occurrence of a helical element (H, H') in a sequence, the matched occurrences of H and H' must be complementary, i.e. to form pairs character by character. To respect how a motif folds, we have to take the matched occurrence of H' in reverse order, thus the first matched character of H must pair with (be complementary to) the last matched character of H', and so forth. The relation "to be complementary" is expressed by a complementarity function $COMP$.

Complementarity function $COMP$ is the same for all helical elements, however may differ for different instances of the problem. $COMP(a, b) = 1$ if $a, b \in \Sigma$ are considered to be complementary in the problem instance. Otherwise, $COMP(a, b) = 0$.

Structural motif M can now be defined as an ordered sequence of single-stranded and helical elements. The position of an element in the ordered sequence corresponds to its position in the motif. Between the two strands of a helical element, there can be arbitrary number of single-stranded elements as well as parts or whole other helical elements. This allows arbitrary pseudoknotted structures. For example $M = (H_1, S_1, H_2, S_3, H'_1, S_4, H'_2)$ would correspond[1] to the motif represented in the Figure 1.2, starting from 5' beginning and ending at 3'. Another example of RNA motif is shown in the Figure 2.1.

If we are speaking about *sequence constraints* of a motif M, we refer to the concatenation of sequence constraints of all the elements which form the motif M.

2.2 Structural Motif Search Problem

We say that a structural motif M (in context of a complementarity function $COMP$) *occurs* in a sequence A, if and only if there exists an *partial increasing function* φ of the sequence constraint indices of motif $M = m_1 \dots m_d$ to the indices in sequence $A = a_1 \dots a_e$, that satisfies:

(i) the image of function φ is one continuous subinterval of indices in A

(ii) only wildcards '*' may be omitted, i.e.

$$\forall i : \ \varphi(i) = \perp \ \Rightarrow \ m_i = \{*\}$$

(iii) every sequence constraint from M fits the character to which it is projected, i.e.

$$\forall m_i \in M : \ \varphi(i) \neq \perp \ \Rightarrow \ FIT(m_i, m_{\varphi(i)}) = 1$$

(iv) for every helical element (H, H') from M:

(a) let $A_{\varphi(H)} = p_1 \dots p_l$ be the string of characters from A to which H is projected in φ

[1] further specification of the elements is required

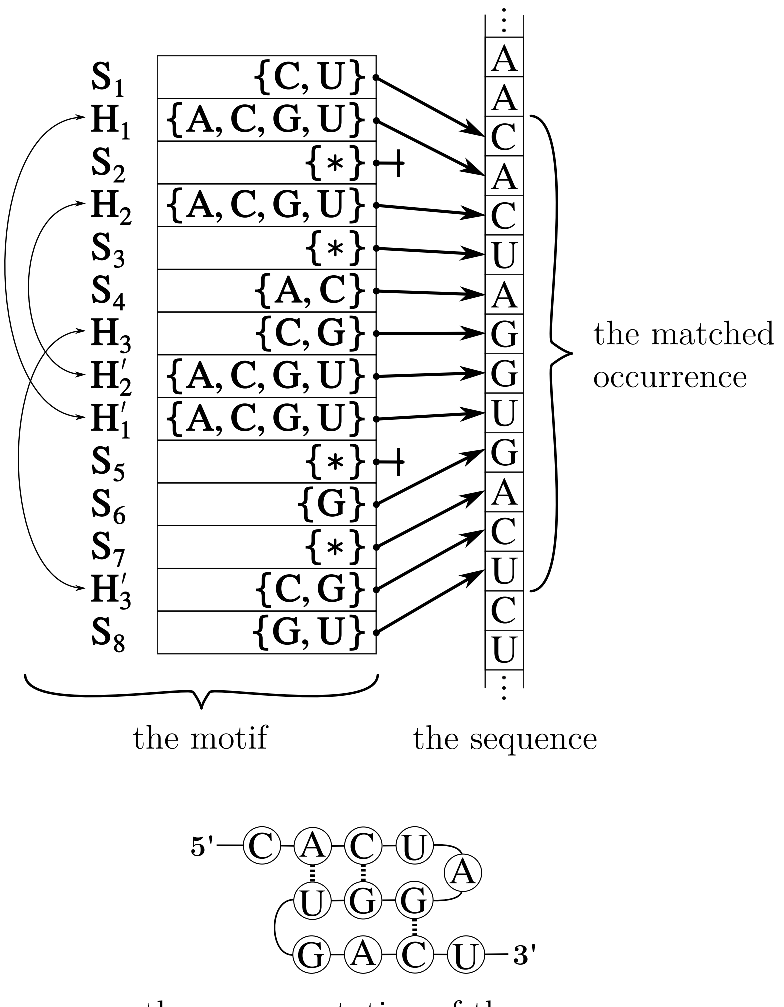


Figure 2.1: An example of RNA motif and of its occurrence in a sequence. For simplicity, the elements are only one character long. The thick arrows represent a matching that defines the motif occurrence.

(b) denote $A_{\varphi(H')} = q_1 \dots q_k$ similarly

(c) $A_{\varphi(H)}$ and $A_{\varphi(H')}$ must be of the same length, i.e. $l = k$

(d) $A_{\varphi(H)}$ and $A_{\varphi(H')}$ must be complementary character by character in terms of complementarity given by the function $COMP$, i.e.

$$COMP(p_i, q_{k-i+1}) = 1 \quad \text{for} \quad i = 1, 2, \dots, k$$

Structural Motif Search Problem (in abbreviation *SMS*) is then a decision problem defined as follows:

> *SMS*: For given alphabet Σ, complementarity function $COMP$, structural motif M and sequence A decide, whether M occurs in A, i.e. whether there exists a correct projection φ of M to A.

2.3 RNA Motif Search Problem

The RNA motif search problem is a special case of SMS problem over RNA alphabet $\Sigma_{RNA} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$, with complementarity function $COMP_{RNA}$ allowing only the canonical base pairs (C-G and A-U). We name this problem *RNA-SMS*.

2.4 NP-completeness of RNA-SMS

Similarly to the previous section, we will first discuss the SMS problem, as it is more general and allows us to perform the proof more illustratively. Afterwords we show how to modify the proof to prove NP-completeness of the RNA-SMS problem.

Our proof of NP-completeness of SMS is inspired by [Lathrop, 1994] where the author proves Protein Threading Problem to be NP-complete.

2.4.1 SMS is NP-complete

It is easy to see that SMS is in NP. For an SMS instance $\Sigma, A, M, COMP$, we can nondeterministically guess a correct occurrence of the structural motif M in the sequence A, i.e. a projection φ of M to A, and then check its correctness in polynomial time.

We will prove NP-hardness by a reduction of ONE-IN-THREE 3SAT to SMS. ONE-IN-THREE 3SAT is known to be NP-complete [Schaefer, 1978]. It is a variant of the 3SAT problem with a restriction that every clause must be satisfied by exactly one literal. Our goal is to construct an encoding from ONE-IN-THREE 3SAT to SMS, i.e. to construct an equivalent instance of SMS for every instance of ONE-IN-THREE 3SAT. A solution to the equivalent SMS instance, i.e. a mapping of a motif M to a sequence A, encodes a solution to the original ONE-IN-THREE 3SAT instance, i.e. a boolean assignment to the variables such that the formula is one-in-three satisfied.

Let us consider a ONE-IN-THREE 3SAT instance: a formula I in conjunctive normal form, where each clause is composed of exactly three literals. We will construct an equivalent instance of SMS: $\Sigma, A, M, COMP$, as follows.

(i) Let the alphabet be $\Sigma = \{\text{T,F,\#}\}$, where T will denote TRUE, F FALSE, and # will serve as a delimiter.

(ii) Sequence A will consist of two parts, the first one dedicated to the composition of the formula I, and the second to an encoding of assignment to the boolean variables of I.

(a) For every clause we construct three gadgets. Each gadget is a triplet of characters from {T,F} and expresses the assignment to the boolean variables of the clause, such that the clause is satisfied by exactly one literal. Since there are three such assignments, we use three gadgets per clause.

The three gadgets are concatenated using # as a separator. All these gadget triplets are also concatenated together, using # as a separator, to form the first part of the sequence A.

For example a clause $(a \vee \neg b \vee c)$ would be encoded by the string TTF#FFF#FTT##, expressing assignments $(a = 1, b = 1, c = 0)$, $(a = 0, b = 0, c = 0)$, and $(a = 0, b = 1, c = 1)$, respectively.

(b) For every boolean variable in the formula I we construct a gadget that will be used to enforce consistent assignment to all occurrences of the given variable. This gadget is simply $\text{T}^n\#\text{F}^n\#$, where n is the number of occurrences of the variable in I.

These gadgets are then concatenated, using # as a separator, to form the second part of the sequence A.

(iii) The complementarity function $COMP$ is defined as follows:

$$COMP(a, b) = \begin{cases} 1 & \textit{iff } a = b \\ 0 & \textit{otherwise} \end{cases}$$

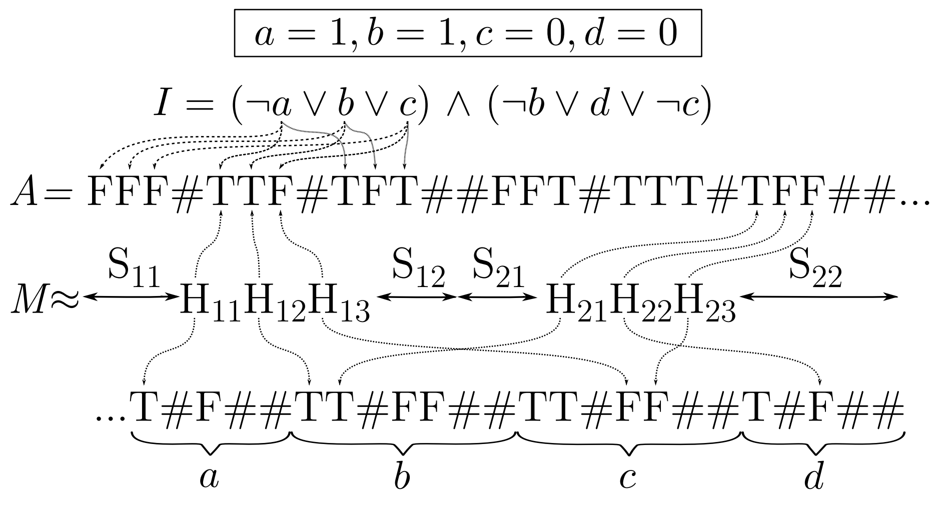
(iv) Finally, we will show how to construct the structural motif M. The job of the motif M is to join satisfying assignments from the first part of A with consistent boolean assignments from the second part. For this purpose helical (paired) elements will be especially useful.

Similarly to the sequence A, motif M has two major parts. The first part will force the assignment to be one-in-three satisfaction for the formula I and the second part of the motif M will enforce consistency of this assignment. Each paired element in the motif M will have its first strand in the first half of the motif and the complementary strand in the second part.

(a) For every clause we construct two unpaired regions and three paired regions, ordered S_1, H_1, H_2, H_3, S_2. Sequence constraints for the elements are:

$$S_1 = s_1 \dots s_8,\ (\forall j) s_j = \{*\}$$

$$\forall i:\ H_i = s,\ H_i' = s,\ s = \{\text{T}, \text{F}\}$$



$a = 1, b = 1, c = 0, d = 0$
$I = (\neg a \vee b \vee c) \wedge (\neg b \vee d \vee \neg c)$
$A=$ FFF#TTF#TFT##FFT#TTT#TFF##...
$M\approx$
S_{11}
$H_{11}H_{12}H_{13}$
S_{12}
S_{21}
$H_{21}H_{22}H_{23}$
S_{22}
...T#F##TT#FF##TT#FF##T#F##
a
b
c
d

$$M = S_{11}, H_{11}, H_{12}, H_{13}, S_{12}, S_{21}, H_{21}, H_{22}, H_{23}, S_{22},$$
$$S_{31}, H'_{11}, S_{32}, S_{41}, H'_{12}, H'_{21}, S_{42},$$
$$S_{51}, H'_{13}, H'_{23}, S_{52}, S_{61}, H'_{22}, S_{62}$$

Figure 2.2: An example of the reduction. The figure shows a formula I, corresponding sequence A, and motif M. For better illustration the motif M is pictured also in a schematic way, to demonstrate how an occurrence of the motif encodes an assignment to boolean variables of I. The illustrated assignment is shown in the box in top part of the figure.

$$S_2 = s_1 \dots s_8 s_9 s_{10},$$
$$(1 \le j \le 8)\ s_j = \{*\},\ s_9 = \{\#\},\ s_{10} = \{\#\}$$

The triplet H_1, H_2, H_3 is intended to match in sequence A one of the three possible assignments to the three variables forming the clause.

The two single stranded elements S_1 and S_2 are of variable length and allow the triplet H_1, H_2, H_3 to have enough freedom to be fitted to any of the three possible assignments encoded in A. In addition, the element S_2 ensures that the triplet cannot be moved any farther, as it requires ## in the end.

(b) When constructing the first part of the motif, for every clause we left behind the complementary parts of helix elements H'_1, H'_2, H'_3. We group all these H'_i by the variable that is used in the i-th literal of the corresponding clause. For a variable x with k occurrences in formula I, we create a submotif starting with a single stranded element S_{x1}, continuing with all H'_i elements corresponding to the variable x and ending with a single stranded element S_{x2}. Sequence constraints for S_{x1} and S_{x2} are:

$$S_{x1} = s_1 \dots s_{k+1},\ (\forall j) s_j = \{*\}$$

$$S_{x2} = s_1 \dots s_{k+1} s_{k+2} s_{k+3},$$
$$(1 \le j \le k+1)\ s_j = \{*\},$$
$$s_{k+2} = \{\#\},\ s_{k+3} = \{\#\}$$

Note that S_{x2} requires ## in the end. By this construction we enforce that the entire group of H'_i matches the block of Ts or the block of Fs in the sequence A. This denotes the assignment of TRUE and FALSE, respectively, to the variable x.

Recall, that in a valid occurrence of a paired element (H_j, H'_j) must both H_j and H'_j match the same character in A. Since the first parts of paired elements can only match a one-in-three satisfiable assignment of a clause and the second parts corresponding to a variable x must all match the same boolean value, then a correct occurrence of M indeed encodes a one-in-three satisfaction of the formula I.

An example construction is shown in Figure 2.2.

In the instance of SMS, that we have just constructed, M occurs in A if and only if the original formula is in ONE-IN-THREE 3SAT.

The reduction is polynomial, since the construction of sequence A is linear[2] in the size of formula I, as well as the construction of motif M. Thus, if we could solve SMS in polynomial time, then we could solve also ONE-IN-THREE 3SAT in polynomial time. Since ONE-IN-THREE 3SAT is NP-complete, SMS is NP-complete, too.

2.4.2 RNA-SMS is NP-complete

The proof of RNA-SMS NP-completeness is analogous. Denote the corresponding sequence A' and motif M'.

[2]For grouping all H'_i by the corresponding variable we can use counting sort, that has linear complexity.

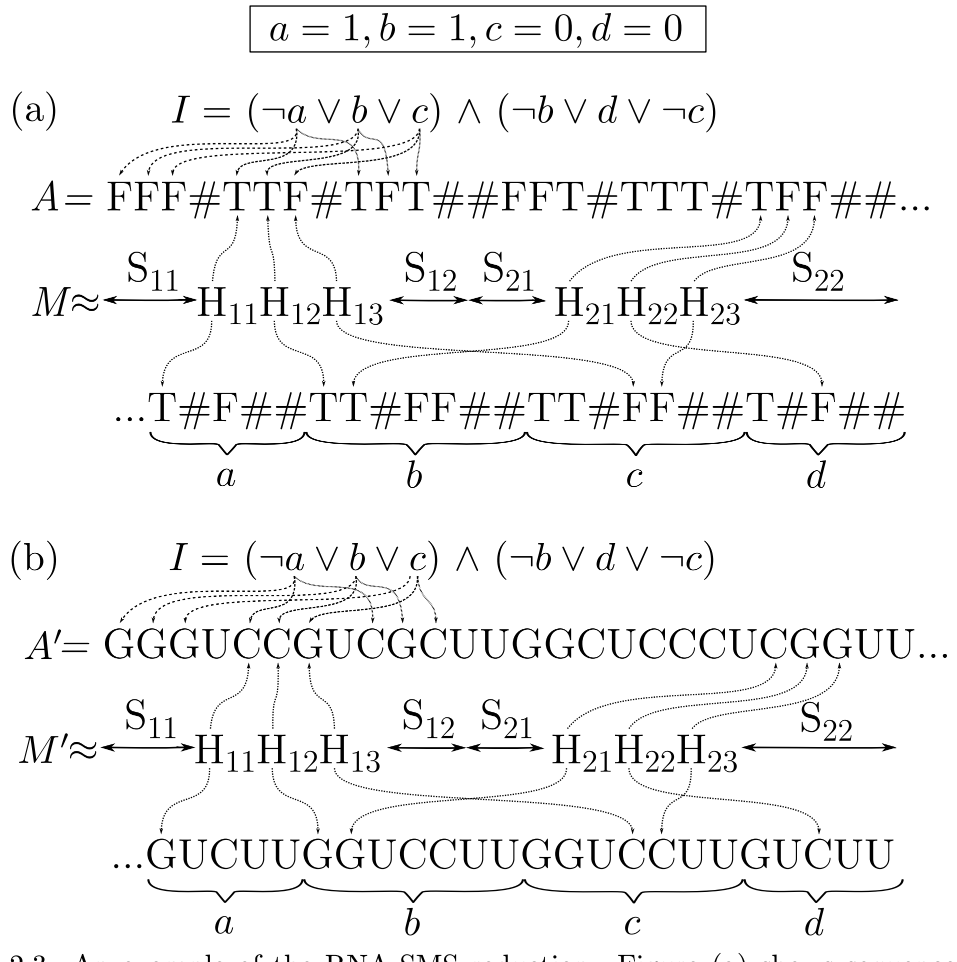

a = 1, b = 1, c = 0, d = 0
(a)
I = (¬a ∨ b ∨ c) ∧ (¬b ∨ d ∨ ¬c)
A= FFF#TTF#TFT##FFT#TTT#TFF##...
S11
S12
S21
S22
M≈
H11H12H13
H21H22H23
...T#F##TT#FF##TT#FF##T#F##
a
b
c
d
(b)
I = (¬a ∨ b ∨ c) ∧ (¬b ∨ d ∨ ¬c)
A′= GGGUCCGUCGCUUGGCUCCCUCGGUU...
S11
S12
S21
S22
M′≈
H11H12H13
H21H22H23
...GUCUUGGUCCUUGGUCCUUGUCUU
a
b
c
d

Figure 2.3: An example of the RNA-SMS reduction. Figure (a) shows sequence A and a sketch of motif M, both constructed in the original way from the SMS proof. Part (b) shows sequence A' and a sketch of motif M' which form an instance of RNA-SMS equivalent to formula I.

The construction of A' and M' is the same as for SMS, with the following adjustments.

(i) In construction of motif M', we change {T,F} to {C,G} and {#} to {U} everywhere it is applicable.

(ii) In the construction of sequence A' we substitute character # for character `U`. When encoding a possible one-in-three satisfiable assignment to variables of a clause, we encode TRUE as `C` and FALSE as `G`. In the second part of A', where gadgets for assignment consistency enforcement are located, we switch the symbols, encoding TRUE as `G` and FALSE as `C`. This switch is necessary as the function $COMP$ in the SMS proof allows only pairs composed of the same two characters, however $COMP_{RNA}$ allows only `C-G`, `G-C`, `A-U`, and `U-A` pairs. Since every paired element from M' has the first strand in the first part of A' and the second strand in the second part of A', this construction of A' is correct.

Figure 2.3 illustrates these changes and compares it to the original encoding used in the SMS proof.

Notice that the reduction remains polynomial. Therefore, we prove that RNA structural motif search problem is NP-complete.

2.5 Discussion

Note that the proof described above could be also done by reduction from standard 3SAT. The main idea of encoding would remain the same. However, for every clause in the formula, we would construct seven, instead of three, gadgets encoding the satisfactory assignments to the variables of the clause. This is because in 3SAT a clause has the condition that the three literals must not be all FALSE at the same time. The number of all satisfactory assignments is then at most $2^3 - 1 = 7$.

Further, we identified two critical motif properties that govern the computational complexity of RNA structural motif search: (i) pseudoknots, and (ii) structural elements of variable length. Leaving out one of these properties would lead to an efficient search algorithm.

If we left out pseudoknots, we could use techniques of dynamic programming to proceed from short parts of the motif to the longer. This approach is similar to the RNA secondary structure prediction algorithm discussed in [Durbin et al., 1998] and leads to polynomial time complexity.

On the other hand, allowing only structural elements of fixed length, we could check each position in a sequence A for an occurrence of the motif M in linear time by a simple check of all primary and secondary constraints of M. Recall that the number of the constraints is linear in length of M. This approach would lead to quadratic time complexity in the length of A.

Chapter 3

RNArobo Algorithm

In the following, we recapitulate the search algorithm implemented in a software tool RNArobo. This algorithm was first introduced in [Rampášek, 2010] following the work of [Gautheret et al., 1990]. Our implementation was originally named RNAMot2, which we later changed to RNArobo. RNArobo can be used as a part of computational pipeline for discovery of RNA motifs [Jimenez et al., 2012]. In this thesis, we use the name RNArobo for the algorithm as well as for its implementation as a tool. It should be clear from the context which meaning are we referring to.

3.1 Algorithm Outline

An RNA motif is composed of several structural elements. We divide the algorithm into two parts:

(i) searching for occurrences of individual elements by a dynamic programming

(ii) assembling motif occurrences from the element occurrences by a backtracking search, which was first published as "Simple Scan" [Gautheret et al., 1990]

The approach is rather straightforward. Assume we search for a motif composed of elements $e_1, e_2, \ldots, e_n$. First, we select element e_1 and search for its occurrences in the sequence. Once we have found one, we continue to search in its close neighborhood for element e_2. If there are multiple e_2 occurrences, we fix the first one, and continue with the element e_3, and so further. This way we try to find a complete occurrence of the motif. In case element e_i could not have been found in the corresponding neighborhood (or we have already tried all its occurrences), we backtrack, and try the next occurrence of e_{i-1}. The neighborhood to be searched for an individual element is determined in a way that ensures consistency of the overall motif match. An illustration of the search procedure is depicted in Figure 3.1. Note, that quality of the search does not depend on a chosen ordering of elements in the search (we could use any permutation), however it affects the running time in practice.

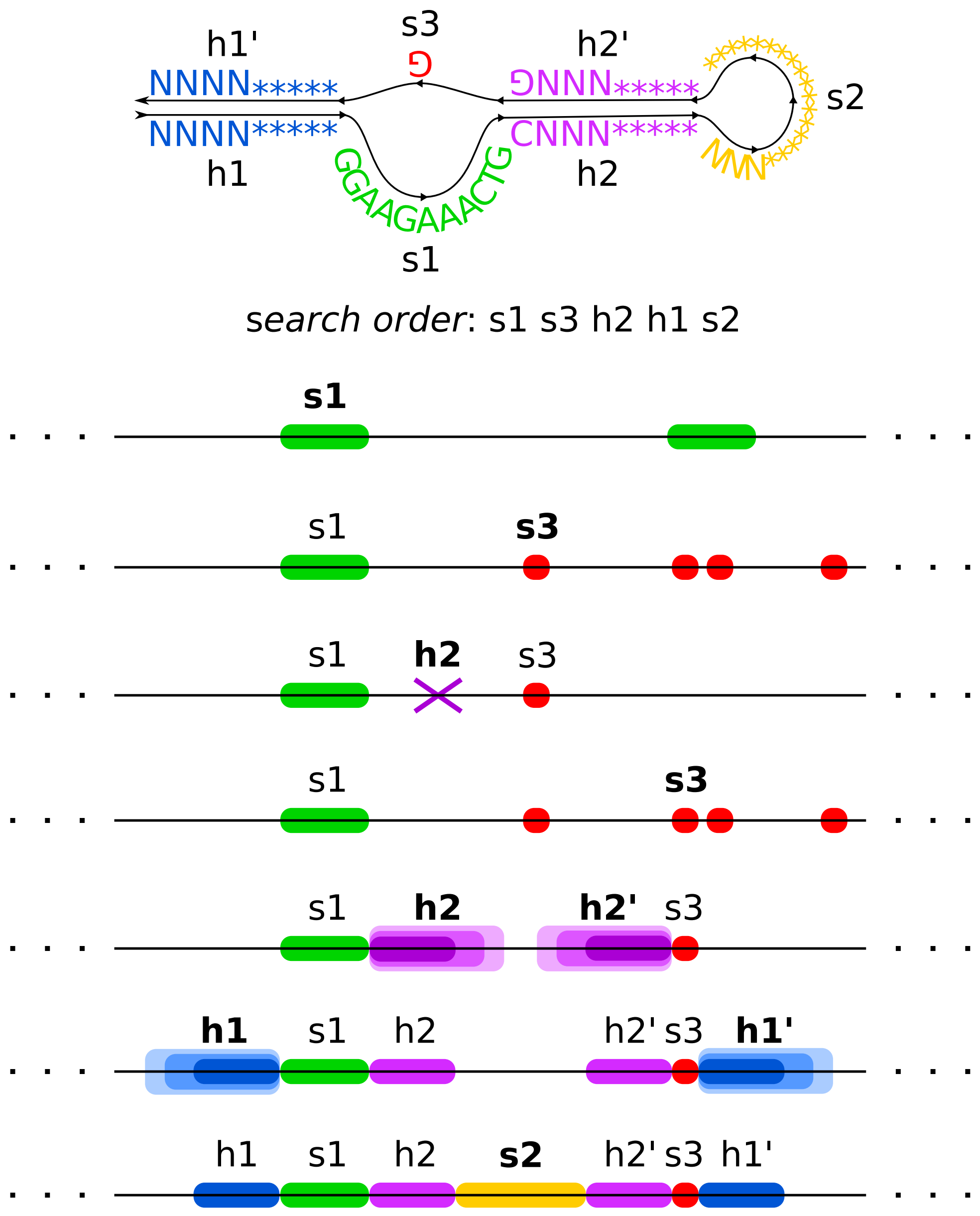


Figure 3.1: An illustration of RNArobo search procedure for a motif of ATP aptamer. The search follows the order of elements `s1`, `s3`, `h2`, `h1`, `s2`.

```
h1 s1 r2 s2 r2' s3 h3 s4 h3' s5 h4 s6 h4' h1' s7

h1 0:2   NNNNNNN:NNNNNNN
r2 0:1   *NNN:NNN* TGCA
h3 0:1:1 NNNNN:NNNNN:A
h4 0:1:1 NNNNN:NNNNN:R
s1 0 TN
s2 0 NNNN[10]
s3 0 N
s4 1 NNSGYN*
s5 0 NN[20]
s6 0 TTC****
s7 0:1 NCCA:A

R s7 h4 s6 h3 h1 r2 s1 s4 s3 s2 s5
```

Listing 3.1: Modified descriptor of a tRNA cloverleaf to demonstrate the descriptor syntax. The first line of the descriptor is the motif map, then specifications of individual elements follow, and in the last line is a command defining the order of elements in the RNArobo search.

3.2 Descriptor Format

Let us now proceed with the specification of the input format of an RNA motif, called a *descriptor*. RNArobo uses a slightly augmented descriptor format of RNAbob [Eddy, 1996]. Thanks to this, RNArobo is compatible with RNAbob descriptors. An example of RNArobo descriptor is shown in Listing 3.1.

A descriptor consists of two main parts:

(i) a *motif map* – a list of structural elements, which establishes the motif composition

(ii) a detailed specification for each of the structural elements

In the motif map, we enumerate the elements in such ordering, as they occur in the motif from the beginning to the end. Identifiers of single strand elements are denoted by the prefix `s`, while paired elements use the prefix `h` or `r` (the difference between these two will be clarified shortly). The rest of the identifier may be arbitrary, yet unique, alphanumeric word. Furthermore, since paired elements are composed of two strands, identifiers for both primary and complementary strand must be placed in the motif map. Identifier for the complementary strand is denoted by an apostrophe at the end, i.e. the complementary strand for `h1` is `h1'`. The primary strand must precede its corresponding complementary strand, but we place no other restrictions on the relative order of individual elements in the map. This way, arbitrary pseudoknotted structures are possible.

The second part of the descriptor provides exact specification of all the elements, one per line. For a **single strand element** `sName` of length m, the format is as follows:

$$\texttt{sName}\ M\!:\!I\ \ S_1S_2\ldots S_m\!:\!S_I$$

The sequence constraints for this element are specified by $S_1S_2\ldots S_m$, where S_i is an upper-case latter specifying a set of admissible residues for position i according to the

IUPAC notation[1]. Alternatively, S_i can be a wildcard denoted as `*`, which allows for either zero or one copy of letter `N` at this position (`N` matches all residues). This way, elements of variable length can be specified. To specify a long variable sequence, the user simply writes a number n in square brackets instead of n asterisks (see element `s5` in Listing 3.1). Further, in the specification is defined the number of allowed mismatches M and insertions I in the element. Letter S_I specifies according to the IUPAC notation which nucleotides are allowed as an single-nucleotide insertion (e.g. there is one insertion of `A` allowed in element `s7` in 3.1). The specification of I and S_I is optional, and can be omitted. In that case, none insertion is allowed. If insertions are allowed, we put restrictions on their position. Insertions are forbidden to occur before the first or after the last symbol of the element. Furthermore, adjacent insertions are forbidden as well.

Specification of a **paired element** is analogous:

$$\texttt{hName}\ \ M\!:\!R\!:\!I\ \ S_1 S_2 \ldots S_m \!:\! S'_m S'_{m-1} \ldots S'_1 \!:\! S_I$$

Compared to single strand specification, we add sequence constraints $S'_m \ldots S'_1$ for the complementary strand, and the number of mispairs R. Similarly to single strand elements, the insertions are not allowed at the beginning and after the end of the element. Further, insertions must not be adjacent nor opposite in context of the paired element. This is a so-called *helical* element, as it has the prefix `h`. We mentioned earlier that another kind of paired element is also possible. It is a so-called *relational* element denoted by a prefix `r`. Relational elements are generalized helical elements, where we can specify an arbitrary complementarity function by a *transformation matrix*. A transformation matrix specifies the rule, according to which bases complementary to `A`-`C`-`G`-`T` are determined. For example, if we want to allow for canonical, and `G`-`T` pairings, the corresponding transformation matrix is `TGYR` (`Y` stands for `T` or `C`, and `R` stands for `T` or `A`). This `TGYR` transformation matrix is implicit for helical elements. The following relational element specification is then equivalent to the previous helical element:

$$\texttt{rName}\ \ M\!:\!R\!:\!I\ \ S_1 S_2 \ldots S_m \!:\! S'_m S'_{m-1} \ldots S'_1 \!:\! S_I\ \ \texttt{TGYR}$$

Optionally, **element reordering command** can be placed in the last line of the descriptor. In the backtracking search, elements will then be used in this particular ordering. This command has no effect on the quality of RNArobo search, however the element ordering can significantly influence the performance. We devote Chapter 4 to the problematic of finding an ordering that leads to short execution time.

3.3 Dynamic Programming for Single Strand Elements

In this section we describe a dynamic programming algorithm which finds all occurrences of a single strand pattern P in a text T with at most M mismatches and I insertions of a one-letter-long pattern P_I.

We use four dimensions of a five-dimensional table S to keep track of position in

[1]see Appendix A on the page 53

T, position in P, the number of occurred mismatches, and the number of insertions, respectively. The fifth dimension is binary, and is intended to serve as a flag, whether the previous aligned symbol of T is an insertion, as one insertion cannot follow another.

Formally, we define a function S as follows:

$$S_{t,p,m,i,b} \in \{0,1\};$$

$$t \in \{0 \dots |T|\}, p \in \{0 \dots |P|\}, m \in \{0 \dots M\}, i \in \{0 \dots I\}, b \in \{0,1\}$$

$$S_{t,p,m,i,b} = \begin{cases} 1 & \textit{iff } P[1\dots p] \text{ can be aligned with a suffix of } T[1\dots t] \\ & \text{with } m \text{ mismatches, and } i \text{ insertions;} \\ & \textit{if } b=1 \textit{ then } T[t] \text{ is an insertion} \\ & \\ 0 & \textit{otherwise} \end{cases}$$

Recurrence

In [Rampášek, 2010] we proposed the following recurrence to compute the function S.

Initial conditions: $\forall t \in \{0 \dots |T|\}\ S_{t,0,0,0,1} := 1$

The recurrence:

$$S_{t,p,m,i,0} = \bigvee \begin{cases} \bigvee_b S_{t-1,p-1,m-x,i,b} & x := (int)(T[t] \text{ does not fit } P[p]) \\ & \\ S_{t,p-1,m,i,0} & \textit{iff } P[p] = \text{`}*\text{' (skip a wild card)} \end{cases}$$

$$S_{t,p,m,i,1} = \bigvee \begin{cases} S_{t-1,p,m,i-1,0} & \textit{iff } T[t] \text{ fits } P_I \text{ (an insertion)} \\ & \\ S_{t,p-1,m,i,1} & \textit{iff } P[p] = \text{`}*\text{' (skip a wildcard)} \end{cases}$$

Solutions:

A match of the pattern P is found in the text T ending at position $t \leq |T|$ with $m \leq M$ mismatches and $i \leq I$ insertions if $S_{t,|P|,m,i,0} = 1$. To obtain this match we have to trace back.

3.4 Dynamic Programming for Paired Elements

The problem in this case is to find all occurrences of a paired pattern $P : P'$ where P, P' are patterns of individual strands (i.e. $|P| = |P'|$) in a text T, such that in an admissible match the individual strands are complementary.

Furthermore, we allow for imperfect matches with up to M mismatches, R mispairings, and with at most I insertions of a one-letter-long pattern P_I together in both strands.

To address this pattern matching problem, we proposed in [Rampášek, 2010] a function H, and a recurrence formula for its computation.

The function H for paired (helical) elements is the following:

$$H_{t_1,t_2,p,m,r,i,b} \in \{0,1\};$$

$$t_1, t_2 \in \{0 \dots |T|\}, p \in \{0 \dots |P|\}, m \in \{0 \dots M\}, r \in \{0 \dots R\}, i \in \{0 \dots I\}, b \in \{0,1\}$$

$$H_{t_1,t_2,p,m,r,i,b} = \begin{cases} 1 & \textit{iff } P[1\dots p] \text{ can be aligned with a suffix } T' \text{ of } T[1\dots t_1] \text{ with} \\ & m \text{ mismatches, } P'[1\dots p] \text{ can be aligned with a prefix } T'' \\ & \text{of } T[t_2 \dots |T|] \text{ with no mismatch, } T' \text{ and } T'' \text{ contain together} \\ & i \text{ insertions, and between } T' \text{ and } T'' \text{ are } r \text{ mispairings;} \\ & \textit{if } b = 1 \textit{ then} \text{ exactly one of } T[t_1], T[t_2] \text{ is an insertion} \\ & \qquad \textit{else} \text{ none of the } T[t_1] \text{ and } T[t_2] \text{ is an insertion} \\ & \\ 0 & \textit{otherwise} \end{cases}$$

Recurrence

Initial conditions: $\forall t_1, t_2 \in \{0 \dots |T|\}\ H_{t_1,t_2,0,0,0,0,1} := 1$

The recurrence:

$$x := (int)(T[t_1] \text{ does not fit } P[p])$$
$$y := (int)(T[t_2] \text{ is not complement of } T[t_1])$$

$$H_{t_1,t_2,p,m,r,i,0} = \bigvee \begin{cases} \bigvee_b H_{t_1-1,t_2+1,p-1,m-x,r-y,i,b} & \textit{iff } T[t_2] \text{ fits } P'[p] \\ & \\ H_{t_1,t_2,p-1,m,r,i,0} & \textit{iff } P[p] = \text{'}*\text{'}^1 \text{ (skip a wildcard)} \end{cases}$$

$$H_{t_1,t_2,p,m,r,i,1} = \bigvee \begin{cases} H_{t_1-1,t_2,p,m,r,i-1,0} & \textit{iff } T[t_1] \text{ fits } P_I \text{ (an insertion)} \\ & \\ H_{t_1,t_2+1,p,m,r,i-1,0} & \textit{iff } T[t_2] \text{ fits } P_I \text{ (an insertion)} \\ & \\ H_{t_1,t_2,p-1,m,r,i,1} & \textit{iff } P[p] = \text{'}*\text{'}^1 \text{ (skip a wildcard)} \end{cases}$$

Solutions:

A match of the pattern $P : P'$ is found in the text T, P ending at position $t_1 \le |T|$, P' beginning at position $t_2 \le |T|$ with $m \le M$ mismatches, $r \le R$ mispairs, and $i \le I$ insertions if $H_{t_1,t_2,|P|,m,r,i,0} = 1$. To obtain this match we have to trace back.

[1]In a correct paired pattern holds: $\forall k\ (P[k] = \text{'}*\text{'} \Leftrightarrow P'[k] = \text{'}*\text{'})$

3.5 Implementation Note

RNArobo is implemented as a C++ console application, and is available for download at `http://compbio.fmph.uniba.sk/rnarobo/`.

We have undertaken several optimizations of the code. Worth mentioning is optimization of the tables used in dynamic programming. These tables are typically sparse, with relatively many zero elements. In the previous versions, we represented elements of the table as arrays of coordinates and stored them in a red-black tree (C++ STL Set). Now, we encode the element coordinates into one 64bit integer. We also tried to replace the red-black trees by hash sets, but this change did not bring intended speed-up in practice (this can change with a better implementation).

Nevertheless, the most important enhancement is a new method for element ordering in the backtracking search, more on this method follows in the next chapter.

For more details on RNArobo algorithm and implementation we refer the reader to [Rampášek, 2010].

Chapter 4

Element Ordering in RNArobo

Ordering of motif elements in the RNArobo backtracking search has a significant impact on the execution time. To design a good element ordering of a complex motif is not a simple task, as many aspects are involved. Therefore we propose a data-driven method for finding a close-to-optimal element ordering, which leads to execution time as short as possible.

4.1 Method Outline

Our approach consists of two main parts:

(i) heuristic proposal of possible orderings

(ii) data-driven evaluation of the proposed orderings

First, we generate all possible starting k-tuples of elements (where k is a parameter). These k-tuples are scored by a heuristic scoring function which takes into account information content of the elements and their relative position within the motif.

All the k-tuples with scores above some threshold are augmented to complete search orderings by the information content heuristic. Then we take random samples from this set and run the motif search with the sampled ordering in a sequence window of a fixed size. For each such application of an ordering, we measure the number of memory operations of the underlying dynamic programming as an approximation of the execution time. Based on the gathered data, we continually eliminate orderings with bad performance. In this way we progress in the search task window by window, but at the same time we observe which orderings lead to the shortest execution times and adapt our strategy.

Additionally, after fixing the first k-tuple of the search ordering, we subsequently use the same technique to find the best successive k-tuple (with respect to the already fixed ordering prefix), and so on, until the whole search order is fixed.

4.2 Heuristic Scoring Function

The main goal of the heuristic evaluation of the proposed k-tuples is to select from all the possible k-tuples a subset, which is small enough to be empirically evaluated. We want this subset to contain at least some k-tuples that can be augmented to a complete element ordering with execution time not far from the optimum. Secondly, we would like not to have many bad tuples in this chosen subset, because as we will see later, their evaluation can increase running time excessively. Naturally, we want this heuristic evaluation to be considerably faster than the empirical.

The score of a k-tuple is a weighted sum of heuristic scores for individual elements of the k-tuple. The weight of an element in this sum decreases exponentially with its distance from the beginning. This is because an element placed sooner in the search order tends to be searched in longer portion of the sequence. Further, the element's score is a linear combination of two heuristic functions h_1 and h_2. Thus the score of a k-tuple $(e^1, \ldots, e^k)$ can be expressed as

$$h(e^1, \ldots, e^k) = \sum_{i=1}^{k} 2^{k-i}\Big(h_1(e^i) + c \cdot h_2(e^1, \ldots, e^i)\Big) \tag{4.1}$$

The first heuristic h_1 is an approximation of the information content of an element, favoring elements that pose more specific constraints. The second heuristic h_2 is order-sensitive and accounts for flexibility of the element's search domain with respect to elements preceding in the ordering. In practice, we use $k = 3$ and $c = -0.3$. After scoring all k-tuples we select those that achieve at least 85% of the maximal score achieved by some of the k-tuples. If there are too many good k-tuples, we limit them to the best 40, however this is a parameter. In the next sections we discuss the heuristic functions in detail.

4.2.1 Information Content Heuristic

By this heuristic, we want to follow the *fail-first*[1] heuristic generally used in backtracking searches [Russell and Norvig, 2010]. This heuristic says that we should search first for the element, which is the least likely to occur. Thus we need to asses the restrictiveness of an element with respect to genomic sequence in which we execute the search. As a measure of restrictiveness, we use *information content* of this element.

Information content is a measure of uncertainty reduction about an outcome once we have received a new piece of information. In other words, it is the difference in the entropy of a random variable before and after some message has been received.

$$I = H_{\text{before}} - H_{\text{after}}$$

Note, that I can have a negative value if after receiving a message the outcome of the random variable is actually more random than we originally expected. Recall, that entropy of a random variable is defined as:

$$H(X) = -\sum_{i} P[X = x_i] \lg P[X = x_i]$$

Here we use logarithm with base 2, so the entropy corresponds to bits.

In our case we assume the initial state to be a uniform distribution over DNA sequences

[1]also called *minimum-remaining-values* or *most-constrained-variable-first* heuristic

of a fixed length. When we are told that an occurrence of an element starts at the first position of a sequence, the distribution is changed from all genomic sequences to those that match the element. Therefore, the information content of an element is entropy of the background genomic sequence distribution minus the entropy of the distribution of sequences that match the given element.

We compute an approximation of the information content, since exact entropy computation of an element with allowed insertions or with variable length is not trivial, and a reasonable approximation is sufficient for our purpose. We approximate the information content of an element as a sum of information contents of its individual positions and then we refine the result according to the number of insertions, mismatches, and mispairs (in paired elements) allowed.

For an **unpaired element**, the background probability distribution of a residue is uniform over the set $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$. Therefore the background entropy $H^u_{\text{before}}(a)$ of an unpaired residue a is:

$$H^u_{\text{before}}(a) = -\sum_{i=1}^{4} \frac{1}{4} \lg \frac{1}{4} = \lg 4 = 2 \text{ bits}$$

Constraints, which an element puts at a particular residue e_i, change the distribution in one of these 5 ways:

1. The element defines a residue e_i without ambiguity, i.e. it allows only one residue from $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$:

   $$H^u_{\text{after}_1}(e_i) = -\lg 1 = 0 \text{ bits}$$

   resulting in information content for such a position to be

   $$I^u_1(e_i) = H^u_{\text{before}}(e_i) - H^u_{\text{after}_1}(e_i) = 2 \text{ bits}$$

2. The residue e_i must be from a subset of size 2 of $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$, in the IUPAC notation[2] this is expressed by one character from $\{\mathtt{M}, \mathtt{R}, \mathtt{W}, \mathtt{S}, \mathtt{Y}, \mathtt{K}\}$:

   $$H^u_{\text{after}_2}(e_i) = -\lg \frac{1}{2} = 1 \text{ bit}$$

   $$I^u_2(e_i) = H^u_{\text{before}}(e_i) - H^u_{\text{after}_2}(e_i) = 2 - 1 = 1 \text{ bit}$$

3. The residue e_i can take one of three specified values from $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$, in the IUPAC notation this is expressed by a character from $\{\mathtt{B}, \mathtt{D}, \mathtt{H}, \mathtt{V}\}$:

   $$H^u_{\text{after}_3}(e_i) = -\lg \frac{1}{3} = 1.585 \text{ bits}$$

   $$I^u_3(e_i) = H^u_{\text{before}}(e_i) - H^u_{\text{after}_3}(e_i) = 2 - 1.585 = 0.415 \text{ bits}$$

4. There is no restriction on the residue e_i (it can take any value), in the IUPAC

[2]see Appendix A

notation this is expressed by character `N`:

$$H^u_{\text{after}_4}(e_i) = -\lg\frac{1}{4} = 2 \text{ bits}$$

$$I^u_4(e_i) = H^u_{\text{before}}(e_i) - H^u_{\text{after}_4}(e_i) = 2 - 2 = 0 \text{ bits}$$

5. Finally, the constraint for e_i can be a wildcard (it can take any value or to be omitted), expressed by character `*`. We assume all the possibilities to be equally likely, therefore:

$$H^u_{\text{after}_5}(e_i) = -\lg\frac{1}{5} = 2.322 \text{ bits}$$

$$I^u_5(e_i) = H^u_{\text{before}}(e_i) - H^u_{\text{after}_5}(e_i) = 2 - 2.322 = -0.322 \text{ bits}$$

   Here we have the situation, when the outcome is actually more random than we have expected.

Information content $I'_{\text{unpaired}}(e)$ of an unpaired element e with no mismatch or insertion allowed is then

$$I'_{\text{unpaired}}(e) = \sum_{i=1}^{|e|}\sum_{j=1}^{5}[e_i \text{ is of type } j]\cdot I^u_j(e_i) \tag{4.2}$$

We will show how to deal with mismatches and insertions later on.

For a **paired element**, we assess its information content as a sum of information contents of the individual pairs that form this element. The background probability distribution of each pair is uniform, hence the entropy $H^p_{\text{before}}(b)$ of a pair b is:

$$H^p_{\text{before}}(b) = -\sum_{i=1}^{4^2}\frac{1}{4^2}\lg\frac{1}{4^2} = -\lg\frac{1}{4^2} = 4 \text{ bits}$$

Let us now show how to compute $H^p_{\text{after}}(e_i)$. The distribution of possible pairs in a paired position e_i is influenced by the sequence constraints of the element, as well as by the constraint that the two residues must be complementary. As mentioned in Chapter 1, in addition to canonical pairs `G-C`, and `A-U`, the couple `G-U` is also often considered to form a pair. Hence in RNArobo we allow a user to define an arbitrary pairing function. Since there are only 16 possible pairs, we can iterate through all of them and verify whether the particular pair satisfies both the conditions. Let us denote the number of such correct pairs r. Additionally, if the pair is `(*,*)`, then there is one more option – to skip this pair (this leads to negative information content for such a pair). Calculation is then straightforward:

$$H^p_{\text{after}}(e_i) = -\sum_{i=1}^{r}\frac{1}{r}\lg\frac{1}{r} = \lg r \text{ bits}$$

resulting in information content of the residue pair e_i to be

$$I^p(e_i) = H^p_{\text{before}}(e_i) - H^p_{\text{after}}(e_i) = 4 - \lg r \text{ bits}$$

Information content $I'_{\text{paired}}(e)$ of an paired element e with no mismatch, mispair, nor insertion allowed is then

$$I'_{\text{paired}}(e) = \sum_{i=1}^{|e|} I^p(e_i) \tag{4.3}$$

In what follows we show how we **approximate the impact of mismatches, mispairs, and insertions** on the information content of an element. Intuitively, these distortions cause more sequences to match an element, i.e. they bring in more uncertainty about the result. Since we do not model them in the background distribution, they will cause negative gain in information content of an element.

Mismatch: A mismatch in an element causes some residue sequence constraint to match everything, i.e. it changes to N. To simulate this, we could find a most constrained position in the element, and subtract its information content from the overall information content. However, we simply subtract 2 bits as a flat rate.

Mispair: A mispair cancels the mutual connection between two paired positions. We could find a pair, whose information content would suffer the most from this loss of pairing. However, we simply take the worst case, i.e. one residue used to unambiguously determine the other. This means loss of 2 bits of information, as one residue of such a former pair does not impose any restrictions on the other residue any more.

Insertion: Insertions are hard to model, as they can occur almost anywhere in the element, and the restriction that no two insertions can be adjacent, nor opposite in paired regions make it even more complicated, as the insertions are not independent. Therefore we decided to model entropy of an insertion directly as the amount of information needed to specify its occurrence, and to neglect their dependency.

The first insertion in an unpaired element e may occur at $m-1$ places, where m is the length of e, and at $2 \cdot (m-1)$ places in a paired element (as it has two strands of length m). The number of positions where a successive insertion may occur differs according to where the preceding insertions occurred. We neglect this fact and consider the number of positions for an insertion to remain constant. To specify where an insertion has occurred, we therefore need $\lg(m-1)$, and $\lg(2m-2)$ bits, respectively. In addition, we need to specify what nucleotide was inserted. To represent this information we need additional 2, $\lg 3$, 1, or zero bits, depending on what nucleotides are allowed for insertion (recall cases in computation of $H^u_{\text{after}_j}$). For example, if only A is allowed to be inserted, we need no additional bits to code this information, as it is implicit.

Now, we can refine previous estimates (4.2), and (4.3), to account also for the mentioned distortions. Let us consider an element e of length m with n_{mm} mismatches, and n_{ins} insertions of x allowed. If e is an unpaired element, its estimated information content is:

$$I_{\text{unpaired}}(e) = I'_{\text{unpaired}}(e) - 2n_{\text{mm}} - n_{\text{ins}}\Big(\lg(m-1) + \sum_{j=1}^{4}[x \text{ is of type } j] \cdot H^u_{\text{after}_j}(x)\Big) \tag{4.4}$$

If e is a paired element with n_{mp} mispairs, then we estimate its information content as:

$$I_{\mathrm{paired}}(e) = I'_{\mathrm{paired}}(e) - 2n_{\mathrm{mm}} - 2n_{\mathrm{mp}} - n_{\mathrm{ins}}\Big(\lg(2m-2) + \sum_{j=1}^{4}[x \text{ is of type } j] \cdot H^{u}_{\mathrm{after}_j}(x)\Big) \tag{4.5}$$

Finally, we define the information content heuristic function $h_1(e)$ as follows:

$$h_1(e) = \begin{cases} I_{\mathrm{unpaired}}(e) & \text{if } e \text{ is an unpaired element} \\ I_{\mathrm{paired}}(e) & \text{if } e \text{ is a paired element} \end{cases} \tag{4.6}$$

4.2.2 Domain Flexibility Heuristic

In RNArobo search, when some element is already fixed and the algorithm tries to find an occurrence of the next element, the position where it should occur is often not determined uniquely. Rather, it has to occur within an interval, which we call the *search domain* of the element. This happens because motif elements may be of variable length. The size of a search domain of an element varies with respect to which elements have been already fixed. The longer the domain is, the more time-consuming is the search for the element occurrence. Furthermore, we are likely to find more occurrences, which we will have to examine individually in backtracking. This heuristic function is meant to reflect these facts.

The input to this heuristic is a non-empty ℓ-tuple, $\ell \leq k$. We assume the first $\ell - 1$ elements to be already fixed by the time the element e^{ℓ} is searched. Of course, we do not known positions of matches of these preceding elements. However, the information that they are already fixed is sufficient to approximate the domain size of the element e^{ℓ}.

For an unpaired element e^{ℓ} we find the nearest fixed element e^{i} (i.e. e^{i} is one of $e^{1}, \ldots, e^{\ell-1}$) on the left side of e^{ℓ}. Then we sum up the flexibilities of all elements between e^{i} and e^{ℓ} in the descriptor. Flexibility of an element is the difference in its maximum and minimum length. We denote this sum F_{left}. The same way we sum on the right side to obtain F_{right}. If there is not a fixed element at some side, we define the corresponding $F_{side} = 0$. Then we approximate the domain size $D^{e^1,\ldots,e^{\ell-1}}_{\mathrm{unpaired}}(e^{\ell})$ as

$$D^{e^1,\ldots,e^{\ell-1}}_{\mathrm{unpaired}}(e^{\ell}) = \min\{F_{\mathrm{left}}, F_{\mathrm{right}}\} + \text{flexibility of } e^{\ell} + 1 \tag{4.7}$$

where we also account for flexibility of the element itself, and plus one for the position present in the domain without any flexibility. Notice that this approximation of the domain size is always positive.

For a paired element e^{ℓ} composed of two strands $e^{\ell}_{\mathrm{first}}$, $e^{\ell}_{\mathrm{second}}$ we proceed similarly. First, we calculate $D^{e^1,\ldots,e^{\ell-1}}_{\mathrm{unpaired}}(e^{\ell}_{\mathrm{first}})$, and $D^{e^1,\ldots,e^{\ell-1}}_{\mathrm{unpaired}}(e^{\ell}_{\mathrm{second}})$ as in (4.7). In this computation of the domain size of one strand, we consider the other strand to be fixed. Since for an occurrence of one strand the complementary strand can occur at any position in its corresponding search domain, the search domain size for e^{ℓ} is the product of domain sizes for individual strands. Thus, we define $D_{\text{paired}}^{e^\ell,\ldots,e^{\ell-1}}(e^\ell)$ for a double strand element as

$$D_{\text{paired}}^{e^1,\ldots,e^{\ell-1}}(e^\ell) = D_{\text{unpaired}}^{e^1,\ldots,e^{\ell-1}}(e^\ell_{\text{first}}) \cdot D_{\text{unpaired}}^{e^1,\ldots,e^{\ell-1}}(e^\ell_{\text{second}}) \tag{4.8}$$

Finally, we define the domain flexibility heuristic function $h_2(e^1, \ldots, e^\ell)$ as follows:

$$h_2(e^1, \ldots, e^\ell) = \begin{cases} D_{\text{unpaired}}^{e^1,\ldots,e^{\ell-1}}(e^\ell) & \text{if } e^\ell \text{ is an unpaired element} \\ D_{\text{paired}}^{e^1,\ldots,e^{\ell-1}}(e^\ell) & \text{if } e^\ell \text{ is a paired element} \end{cases} \tag{4.9}$$

4.3 Candidate Sampling

Recall that the heuristic function defined above is used to select promising initial k-tuples for search orders. These k-tuples are then augmented to complete search orderings by the information content heuristic. Denote the resulting set of complete orderings as O.

From O we uniformly choose independent and identically distributed (i.i.d.) samples. Subsequently we run the motif search with the sampled ordering in a sequence window of a fixed size. For each sample $x \in O$, we measure T_x, the number of memory operations (read and write) of the underlying dynamic programming, as an approximation of the execution time, which is universal for all machines. Each sequence window is used only once. The next window shares an overlap with the previous one, so that the search cannot miss a motif occurrence. This way we progress in the search task window by window, and at the same time we evaluate candidate orderings from O. In practice, we set the window size to be

$$\max\{10 \cdot \text{max length of an occurrence}, 1000\}$$

However, it is not always possible to divide the searched sequences into windows of the same size, especially when the search database contains many short sequences. Therefore we normalize the number of measured memory operations by the actual window size, and we use as T_x the number of memory operations per one base of a window.

We approximate the distribution of the random variable T_x by a Normal distribution with unknown mean and variance. Examples of empirical distribution of T_x can be seen in Figure 6.2 in Chapter 6.

Our goal is to pick, from the set of candidate orderings O, the ordering which leads to the shortest execution time on average. Formally, we want to find $x^* \in O$, such that $\forall y,\ E[T_{x^*}] \leq E[T_y]$. We use Welch's t-test (described in more detail in Section 4.3.1) to test the null hypothesis $E[T_x] \geq E[T_y]$ against the alternative hypothesis $E[T_x] < E[T_y]$. Each time we gather a new sample from T_x for some $x \in O$, we run tests of x against the rest of O. Thus, as soon as we observe statistically significant difference between two candidates, we can immediately eliminate the one with significantly higher mean time of execution from set O. In the case when the mean of two candidates cannot be compared with enough significance, even after both were sampled many times (we use threshold of 100 samples from each), we break the tie. We simply eliminate the one with the higher sampled mean.

In this way we sample and test until all but one ultimate ordering are eliminated, and we have the winning initial k-tuple. Recall, that this initial k-tuple was extended to a full ordering by the information content heuristic. Now, we can drop this extension and start training the following k-tuple in the ordering with already fixed prefix. In this way we prolong the fixed prefix, until we have a completely fixed order.

4.3.1 Details of Welch's t-test

In this section we briefly describe the Welch's t-test [Welch, 1947] used in our candidate elimination.

In statistics, the so-called Behrens-Fisher problem is the problem of hypothesis testing, which concerns difference between the actual means of two normally distributed populations with possibly unequal variances, based on independent sets of samples from these distributions. This problem is a generalization of the Student's problem that assumes the variance of the populations to be equal. We decided to choose a method that can address the Behrens-Fisher problem, because in candidate elimination we cannot assume equality of variances of variables T_x.

A variety of methods has been proposed to address this problem. They differ in robustness against Type I errors under violations of normality, difference in the sample groups sizes, or difference in the variances. For more details we refer the reader to [Kim and Cohen, 1998, Ruxton, 2006]. We decided to use the Welch's t-test as it has the best combination of performance and ease of use [Ruxton, 2006].

Welch's t-test is based on Student's t-test, which assumes the variances of the two populations to be equal. Welch proposed several methods to correct the number of degrees of freedom in Student's t-test in case of unequal variances. Probably the most commonly used approximation is the Welch-Satterthwaite equation, [Satterthwaite, 1946]. Therefore this method is also called Welch-Satterthwaite test, or Smith/Welch/Satterwaite (SWS) test, acknowledging also the work of Smith [Smith, 1936]. However the most commonly used name is simply "unequal variance t-test".

Welch's t-test, similarly to Student's t-test, is comprised of calculating a t statistic that is then compared with the value in standard t tables according to the appropriate number of degrees of freedom ν. The statistic t is defined by the following formula:

$$t = \frac{\mu_1 - \mu_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{4.10}$$

where μ_i is the sample mean, s_i^2 the sample variance, and n_i the sample size of the i^{th} population.

The approximate number of degrees of freedom ν is then obtained form the Welch-Satterthwaite equation:

$$\nu = \frac{(g_1 + g_2)^2}{\frac{g_1^2}{n_1-1} + \frac{g_2^2}{n_2-1}} \qquad \text{where } g_i = \frac{s_i^2}{n_i} \tag{4.11}$$

With this ν computed, we look-up the appropriate critical value of t in a table of t-distribution and execute a one-tailed test. If the computed t is greater than the critical value, we reject the null hypothesis in favour of the alternative hypothesis. In our algorithm we test at significance level $\alpha = 0.025$.

Chapter 5

Post-processing of RNArobo Results

The fact that a piece of sequence matches an RNA structural motif is not sufficient to expect that this sequence really folds into the desired structure. This happens because our search task represents only a very simplified model of the physical forces involved in RNA folding. To at least partially address this problem, we have implemented a set of software tools, which enable a user to sort and filter the found motif occurrences according to estimated structural stability. In this chapter, we use the term *false positive* for a match that is not correct in the context of searching for real homologs. Tools described in this chapter were developed under the guidance of Prof. Andrej Lupták. These tools are intended to mimic bioinformatic analyses that biochemists would do manually to verify the functionality of proposed matches.

5.1 RNA Structure Prediction

Syntactical match of an RNA structural motif in RNArobo search does not imply that this desired structure is the most stable (probable) structure for this found piece of RNA sequence (the match). A natural way to address this issue would be to calculate the most stable structure of this sequence and then compare it to the desired structure.

Here comes the problem. We have to solve the *RNA structure prediction* problem, also referred to as *RNA folding* problem. This problem has been thoroughly studied for past decades, leading to many different algorithms.

These algorithms often differ in definition of the optimal secondary structure they are looking for. Usually, they seek to the structure in which the RNA sequence has the lowest minimum free energy (MFE), which tends to be the most stable structure. The original methods for MFE structures computation typically use combinatorial optimization techniques (e.g. dynamic programming). The most recognized algorithms in this group are Nussinov algorithm [Nussinov and Jacobson, 1980], Zuker-Sankoff algorithm [Zuker and Sankoff, 1984], and McCaskill algorithm [McCaskill, 1990].

However there is evidence that RNA sequences start folding right away during the process of their transcription from DNA. This may cause a sequence to fold into a local optima different from the globally optimal MFE structure. Folding kinetics approaches account for this phenomenon, such as KineFold [Xayaphoummine et al., 2005].

Comparative methods try to exploit known structures of RNA sequences similar to the query sequence. This method is suitable for sequences belonging to a known RNA family. To model the consensus structures of a given family and then to infer a structure for the query sequence, algorithms mainly use stochastic context-free grammars, and their variations (e.g. covariance models).

Despite a lot of effort, accuracy of the exiting general-purpose methods is rather low for more complex structures. However, the main pitfall of RNA structure prediction are pseudoknots, as they cannot be directly modeled by stochastic context-free grammars or by common energy models. In fact, the general pseudoknot prediction has been proven NP-hard [Lyngsø, 2004]. Lately, many methods have been proposed to tractably address the pseudoknot prediction problem in practice also for longer sequences [Ren et al., 2005, Bellaousov and Mathews, 2010, Sperschneider and Datta, 2010, Sato et al., 2011]. Unfortunately, none of the existing general-purpose tools can reliably predict complex structures, like the double pseudoknotted structure of the HDV ribozyme, which is particularly complicated.

Therefore to filter out false positives from RNArobo search results, we cannot simply use an RNA structure prediction tool, as we proposed at the beginning of this section. Besides the problem with the accuracy of the structure prediction, this approach has one more drawback. Comparing the predicted structure to the structure prescribed by a descriptor is not trivial, because the real homologous motif occurrences seldom share exactly the same structure. Slight structural variations in some parts of the motif are permissible, while some parts of the motif are crucial for the functionality, and are much more structurally and sequentially conserved. We try to model these phenomena in the RNArobo search by allowing for mismatches, mispairs, insertions, or wildcards, that cause the search to be more sensitive however less specific. It is therefore crucial to account for this variability or strictness of individual parts of the RNA motif primarily in post-processing phase. Naturally, expert knowledge on the searched RNA motif is necessary for more precise *in silico* evaluation of the candidates, proposed by RNArobo search.

5.2 Evaluation of Structural Stability *per-partes*

Our approach allows a user to select several submotifs of the searched motif. For each such submotif we use external tools for RNA structure prediction and based on the result we assign it some score. The score of a motif occurrence is then a linear combination of submotif scores.

These submotifs do not have to be continuous in the motif, nor they have to be mutually disjunct, i.e. a submotif can be an arbitrary list of structural elements present in the motif. It is advisable to select submotifs, whose structure can be predicted with sufficient accuracy. For a particular motif occurrence, nucleotide sequences belonging to the individual elements are parsed out, and for each submotif the corresponding sequence is assambled together. We then compute a predicted structure (fold) and its minimum free energy (MFE), using ViennaRNA package [Hofacker et al., 1994, Lorenz et al., 2011], and DotKnot [Sperschneider and Datta, 2010] in case of pseudoknotted submotifs. Score of each submotif is then computed by one of the following ways:

(i) stability of the predicted fold, the score is the absolute value of the predicted MFE

(ii) stability of the predicted fold in terms of the predicted MFE per one base

(iii) closure of the predicted fold, the score is the percentage of first n bases of the submotif that pair with last n bases of the submotif

The score of the motif occurrence is then a linear combination of submotif scores. The parameters of the linear function are defined by the user. Recall that the submotifs can be arbitrary, e.g. a user can define the same submotif twice and use different score functions. For example, this could lead to scoring the submotif according to its MFE as well as according to its fold closure at the same time.

We have implemented this method in a tool called FoldFilter. It allows an expert user to assess RNArobo search results, and filter out majority of false positive matches. The best occurrences are then good candidates for further *in vitro* testing.

To facilitate estimation of the submotif weights we introduce a tool called WeightsTool, which allows the user to dynamically change these weights and see how the changes affect the score distribution over all motif occurrences. It is then possible to export all occurrences with score over a desired threshold.

5.3 Implementation Note

Both tools for post-processing of RNArobo results, FoldFilter and WeightsTool, are implemented as Perl scripts. They are available for download at the official RNArobo website: `http://compbio.fmph.uniba.sk/rnarobo/`.

Prerequisites:

- properly installed ViennaRNA package
- DotKnot (provided in the script package)
- Python interpreter to run DotKnot
- Perl interpreter to run FoldFilter and WeightsTool

Functionality of the tools was tested in various Linux and Mac OS environments.

Chapter 6

Experimental Results

In this chapter we present empirical results obtained by running RNArobo on real biological sequences with several realistic descriptors. First we test validity of several assumptions made in Chapter 4. We then compare the overall running time of RNArobo with several existing tools and study the progress of ordering elimination in our data-driven element ordering (DDEO) strategy.

6.1 Used RNA Motifs

In our experiments we used the following four RNA motifs, their descriptors are listed in Appendix B.

- Motif of an *ATP aptamer*: A rather simple motif with a conserved single-stranded element, but also with some elements of variable length. An illustration of the motif is depicted in Figure 3.1.
- Motif of a *Hepatitis Delta Virus ribozyme*'s double pseudoknot: This is the most complex motif in our set. It contains sequentially conserved elements as well as variable elements, and helices with allowed mispairs. But above all, it is the double pseudoknot that makes the motif complicated.
- Motif of a *Hammerhead ribozyme*: In the motif there are single-stranded elements that are sequentially conserved, but also two very variable elements. Helices are also of variable length, but a perfect canonical pairing is required.
- Generalized motif of a *tRNA cloverleaf*: This motif contains structural elements of variable length and helices with allowed mispairs. These properties with lack of sequential conservation slow down the search, and cause the motif to have many matches.

6.2 Used Genomic Sequences

We have used real DNA sequences:

- Genome of a fruit fly (*Drosophila melanogaster*) [NCBI, 2012a]. This database is 168,736,537 bases long.
- Human genome [NCBI, 2012b], that is 3,137,161,264 bases long.

6.3 Hardware and Software Environment

As a machine for our tests, we used a server with eight physical cores (2× quad-core Intel Xeon E5520 @ 2.27GHz), and 16GB of RAM.

The operating system was Ubuntu Server 10.04.4 LTS. For compilation of C/C++ source code we used GCC 4.4.3 compiler with optimization level 3 (`-O3` flag).

6.4 Memory Operations vs. Real Time

As mentioned earlier in Section 4.3, we use the number of memory operations done during dynamic programming as a measure of the real execution time in DDEO. By a memory operation we mean any access to the dynamic programming table, i.e. both read and write operations are counted equally. We have run several experiments to demonstrate that this measure correlates well with the real execution time.

We have chosen 9 different search orders of elements in the tRNA motif. Then with each of these fixed orderings we have run RNArobo search in a fixed ~2.2 million bases long sequence. We measured both execution time and memory operations. In Figure 6.1 we plotted the average number of memory operations per second for each of these experiments. The execution time varied from a few second to several minutes, however the ratio of memory operations per second stays rather steady. Note that the ratio slightly varies due to the influence of all the other operations that are involved, as in these experiments we measured the total execution time, including input parsing, overhead of backtracking search, and so forth.

These results encourage us to use the number of memory operations as a good machine-independent measure of the execution time for the purpose of our data-driven element ordering method.

6.5 Normality of T_x

In our DDEO strategy (Section 4.3) we measure the number of memory operations as a measure of the execution time and then we use such observations to eliminate search orders by a statistical test. Here we test the normality of the variables involved in this test. Recall, the random variable T_x for an order of elements x, is the number of memory operations executed during RNArobo search in a sequence of a fixed size and then normalized by this window size. In other words, T_x is the average number of memory operations per one residue in a sequence window. For the purpose of candidate elimination in Section 4.3

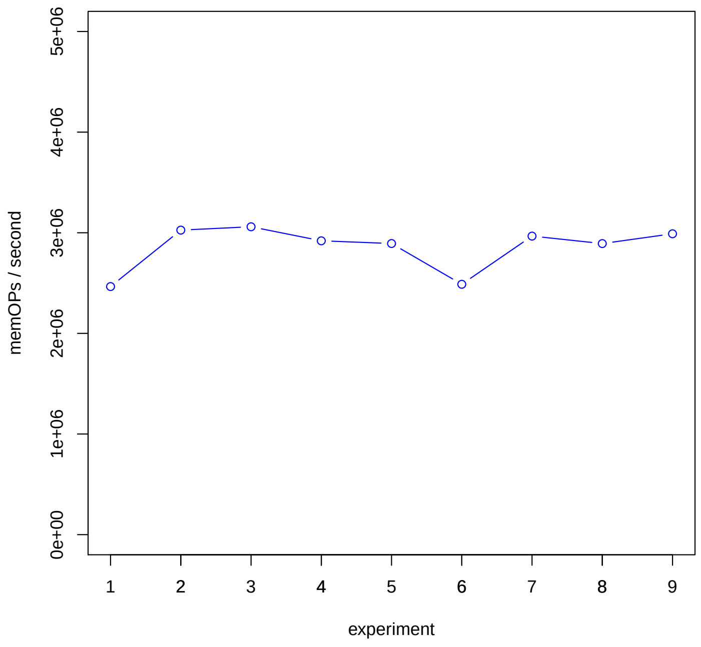


Figure 6.1: The the number of dynamic programming memory operations per one second of the overall execution time in our 9 experiments does not vary significantly.

we assume T_x to have Normal distribution. We want to show that this assumption has rational foundation.

We measured ~530 samples of T_x for four different element orders of the tRNA motif. Although the measured values do not pass statistic normality tests, in Figure 6.2 we can see that the Normal distribution is a good approximation.

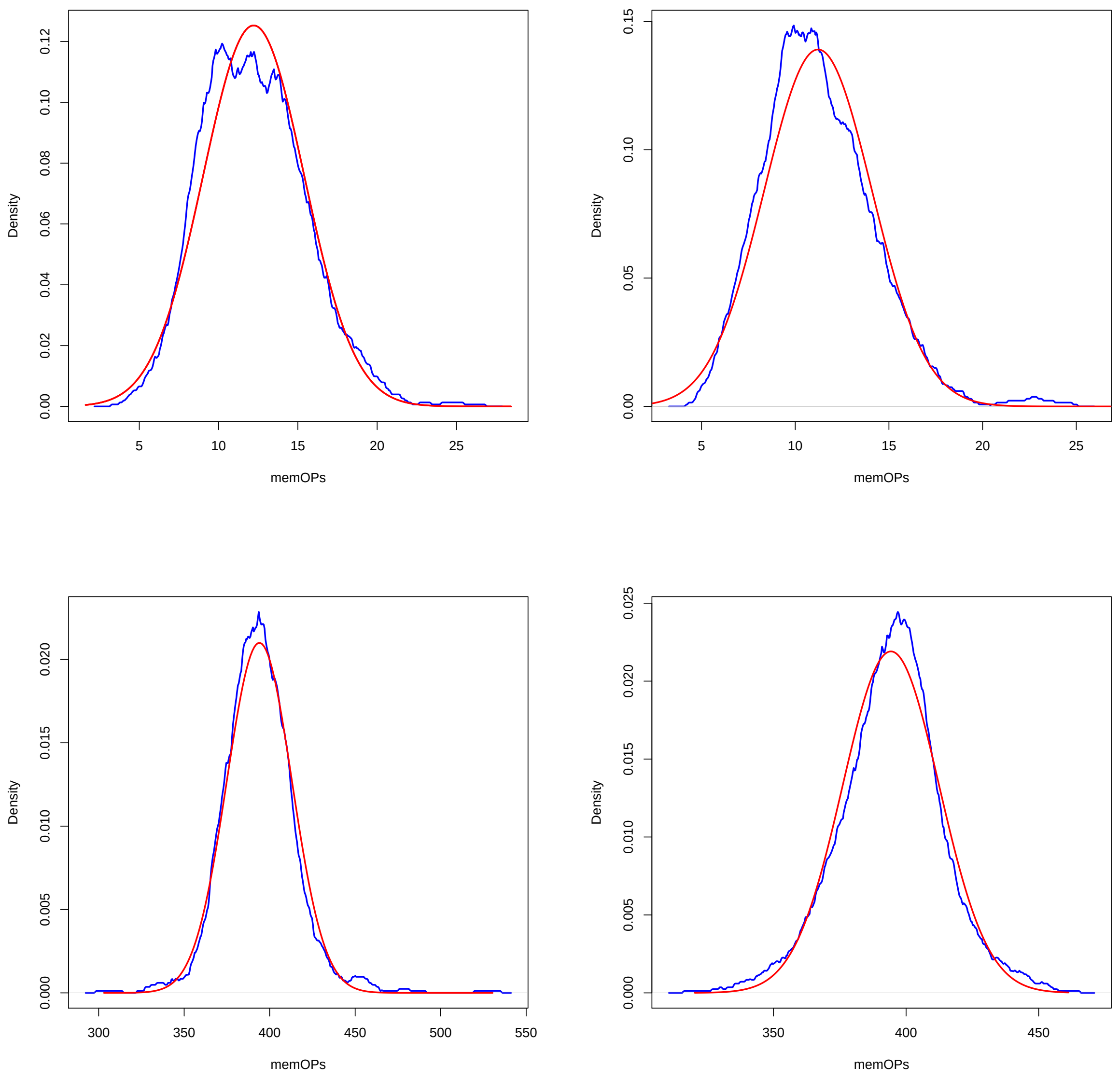


Figure 6.2: Sampled distributions of T_x (the number of memory operations per residue) on different sequence windows using four different fixed orders of elements. In each plot, the blue curve is a kernel density estimation of the T_x samples using uniform (rectangular) kernel. In red is depicted the Normal distribution with parameters set to sample mean and sample variance.

6.6 Execution Time

In this section we compare the running time of RNArobo with data-driven element ordering (RNArobo 2.0) to the previous RNArobo version 1.94, and to established search tools RNAbob 2.2 (1999) [Eddy, 1996], and RNAmotif 3.0.7 (2010) [Macke et al., 2001]. Note, that RNArobo 1.94 uses a fixed search ordering obtained by a simple heuristic, and is otherwise identical to RNArobo 2.0.

We have measured three runs of RNArobo 2.0 (with DDEO), and then run the search again with the best search order found in individual runs. This way we can assess the quality of found orderings as well as the overhead of the DDEO, which fluctuates as randomization is involved.

In DDEO, we use the following parameter values:

- $k = 3$
- cutoff score for k-tuples is 85% of the maximum score
- limit for O is $|O| \leq 40$ (if there is too many good candidates over cutoff threshold)
- window size is $\max\{10 \cdot \text{max length of a motif occurrence}, 1000\}$
- significance level for Welch's t-test is $\alpha = 0.025$

After fixing the first triplet of the search order, we use the same technique to find the best successive triplet, and so further, until the whole search order is fixed.

Elimination of candidates by Welch's t-test at significance level $\alpha = 0.025$ appears to be the best trade-off between stability and execution time. Testing with higher α tends to be less stable. The search converges to a final ordering rather quickly, however mistakes leading to worse ordering are more likely to happen. On the other hand, lower α forces the search to execute more samples, including those with inferior performance.

There may not be one unique ordering that is significantly better than all the others, therefore we break ties between two orderings, after each has more than 100 samples, by eliminating the ordering with the higher sample mean.

We have conducted three sets of tests:

- on chromosome 3 of the Drosophila genome (27,905,053 bases; Figure 6.3)
- on the whole genome of a Drosophila (168,736,537 bases; Figure 6.4)
- on the whole human genome (3,137,161,264 bases; Figure 6.5)

In each test, we scanned both strands of the DNA, thus doubling the effective sequence length. We tested RNArobo 1.94, a version with unsophisticated heuristic for element ordering, only in the first test set. As can be seen in Figure 6.3, this heuristic works well for a simple motif of ATP aptamer, but for more complex structures it fails to deliver a reasonable ordering. Thus we left out RNArobo 1.94 from the genome-wide tests.

In Figure 6.3 we use a relatively short sequence, and for a complex motif, such as Hammerhead ribozyme, RNArobo with DDEO needs a much longer time than RNArobo run with the best order already given at the start. This is due to bad candidate orders, whose empirical evaluation is time-consuming. However notice, that DDEO overhead is roughly constant with respect to the length of the searched genomic sequence. As we increase the sequence length (Figures 6.4 and 6.5), we can see that this overhead is becoming negligible. Therefore in case of search in large database, e.g. genome-wide search, it is advisable to sacrifice longer DDEO overhead in return for better ordering, which eventually leads to shorter execution time.

We can see that RNArobo 2.0 even with DDEO overhead outperforms RNAbob and RNAmotif in case of complex motifs like HDV pseudoknot and tRNA. In this case RNArobo benefits from the ability to find a close-to-optimal search ordering, and also from its robust underlying dynamic programming. For the other two motifs, RNArobo 2.0 is slightly slower than RNAbob, and RNAmotif performs far the best. Notable is the rather steady performance of RNArobo throughout the whole test set. For example the difference in RNAmotif performance for ATP aptamer and for HDV pseudoknot is enormous, while for RNArobo the difference is fairly small. This shows, that the bottleneck of RNArobo performance is now the dynamic programming used in search for individual element occurrences. RNArobo 2.0 does not use any pre-filtering techniques, or specialized pattern matching algorithms in case there are no mismatches or mispairs allowed in an element, but always uses the general dynamic programming. Thus we believe there is still room for further optimization.

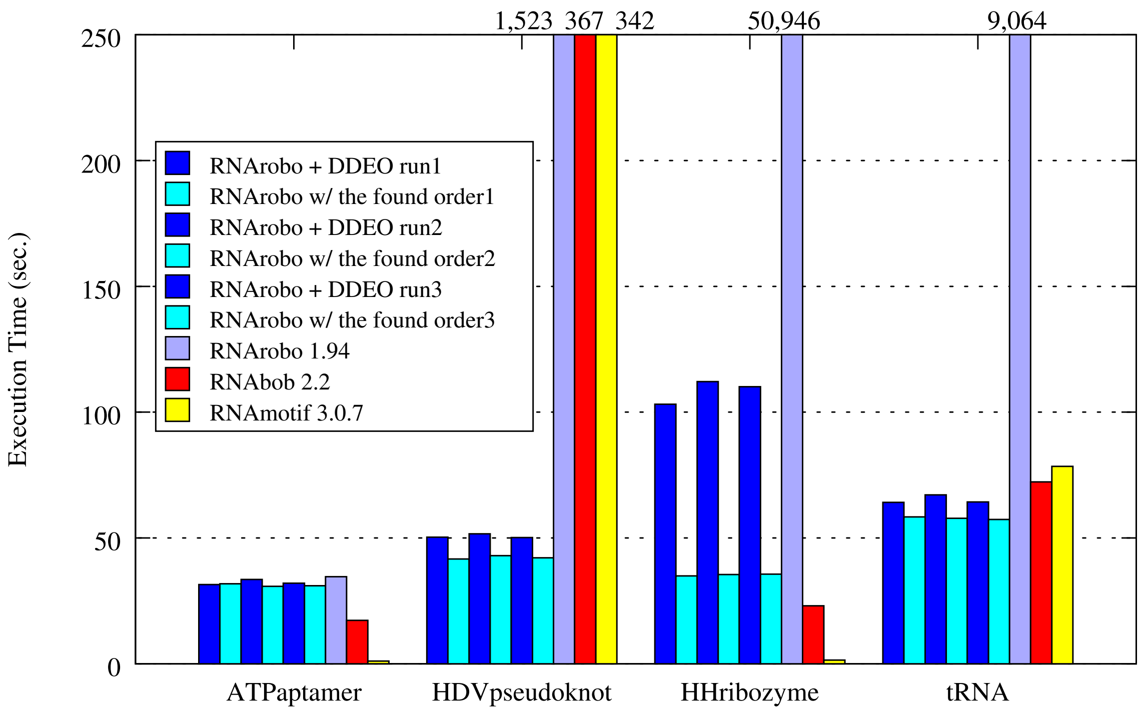


Figure 6.3: Execution time of RNArobo 2.0 (with DDEO) compared to its previous version 1.94, RNAbob 2.2, and RNAmotif 3.0.7. The search was conducted on chromosome 3 of the Drosophila genome (55,810,106 bases scanned). Element ordering heuristic in RNArobo 1.94 fails to find an effective order for more complex structures. New data-driven method in RNArobo 2.0 works well, however the overhead is notable as the sequence is short.

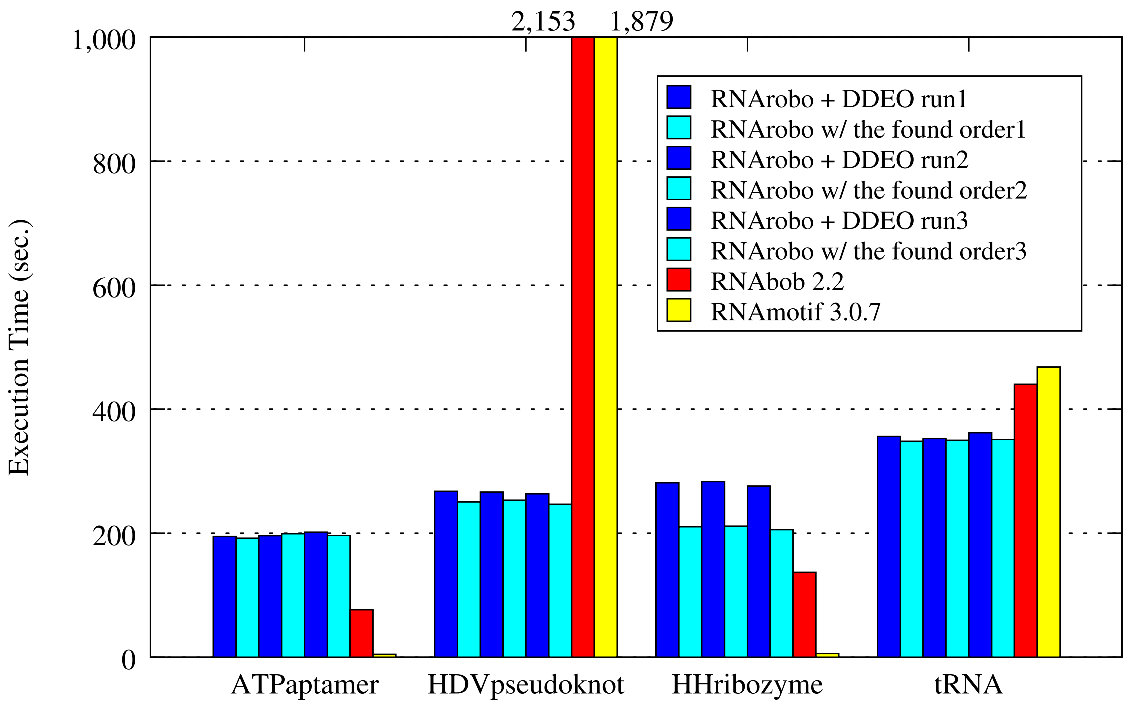


Figure 6.4: Search in the genome of Drosophila (337,473,074 bases scanned). We compare RNArobo 2.0 to RNArobo with fixed previously found order, RNAbob 2.2, and RNAmotif 3.0.7.

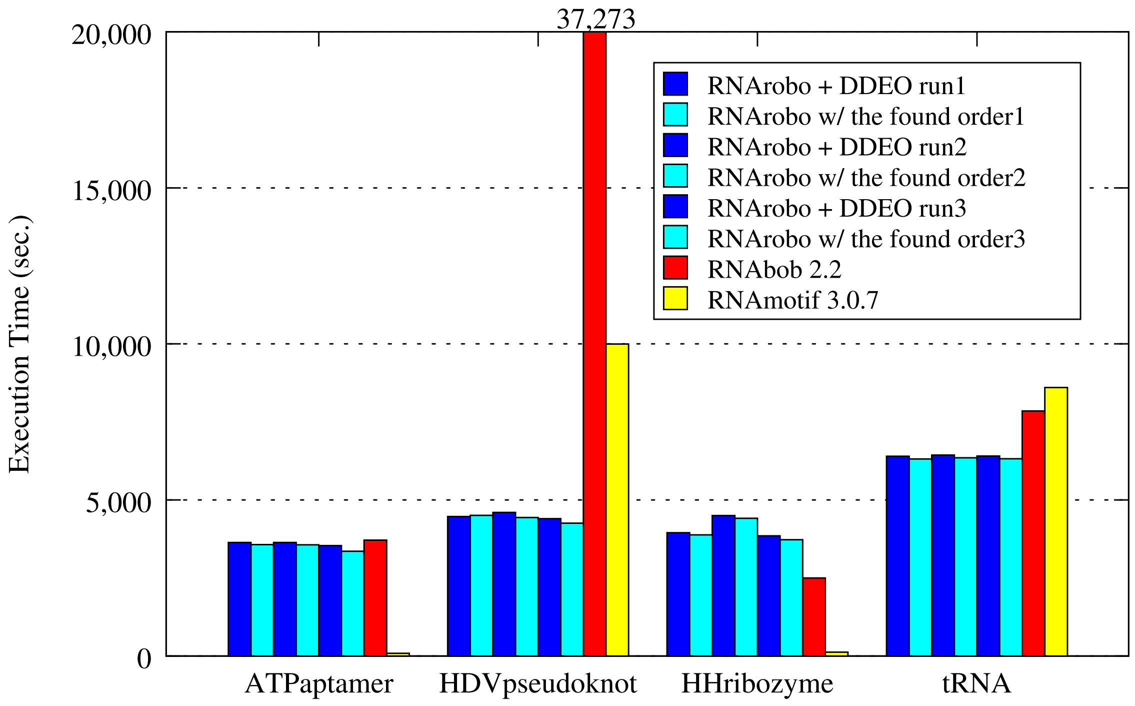


Figure 6.5: Execution time for the whole human genome search (6,274,322,528 bases scanned). The overhead of data-driven element ordering method in RNArobo 2.0 is negligible for long databases.

6.7 Details of DDEO Operation

We devote this section to the analysis of operation of our data-driven element ordering (DDEO) method proposed in Chapter 4 and implemented in RNArobo 2.0. During the search for four motifs in the human genome described in the previous section, we recorded statistics for all sampled candidates. Based on this data we want to explore how does the DDEO method progress in practice. The overall execution time of these searches can be seen in Figure 6.5.

For each of the four motifs we present two types of figures. The first type (e.g. Figure 6.6) depicts the value of T_x averaged over windows where a particular ordering was used, i.e. it is the average number of memory operations per one base of a sampled window. We plotted this value from three runs of RNArobo 2.0 for each of the best k-tuples that were selected to set O (and augmented to complete element orders) based on our heuristic scoring function. Notice, that the heuristic proposal of the k-tuples is deterministic, therefore in each run the k-tuples are the same, however the sampled values from their distribution vary.

In the second type of figures we depict the number of samples done for each of the proposed k-tuples before it was eliminated or chosen as the very best k-tuple. We refer to this number also as the *coverage* of a given k-tuple. In general, we can see the trend that the k-tuples with inferior performance have low coverage, as they are quickly eliminated if there is a large enough proportion of candidates with a good performance (as it is then more likely that we sample a good candidate that will eliminate these inferior ones). On the other hand, the best performing k-tuples with similar values have higher coverage, as more samples are required to eliminate one of them. Recall that we break tie between two k-tuples once we have obtained at least 100 samples of each.

In Figures 6.6 and 6.7 we can see artefacts of the DDEO method procedure in search for the best element ordering of the ATP aptamer motif. The performance of heuristically proposed candidate orders have very similar performance. This causes the DDEO to take many samples, as difference between candidates are not significant in the Welch's t-test.

The other three motifs (Hammerhead ribozyme, HDV pseudoknot, and tRNA) are more complex than ATP aptamer, and are composed of more elements. Therefore we present for them also figures from training of the *second triplet* of their search order.

As we can see in Figures 6.8, 6.14, and 6.20, our heuristic does not work reliably for these motifs. It proposes several search orders with notably bad performance, however there are still enough good candidates that quickly take over. In case of the Hammerhead ribozyme (Figure 6.14), there are four exceptionally inferior candidates proposed. This results in the large overhead noticeable in Figures 6.3, and 6.4.

It is worth mentioning that even though choices of the final k-tuples in individual RNArobo 2.0 runs slightly differ, their overall performance is almost the same. The only exception is the second run of Hammerhead ribozyme in human genome, which is a little worse.

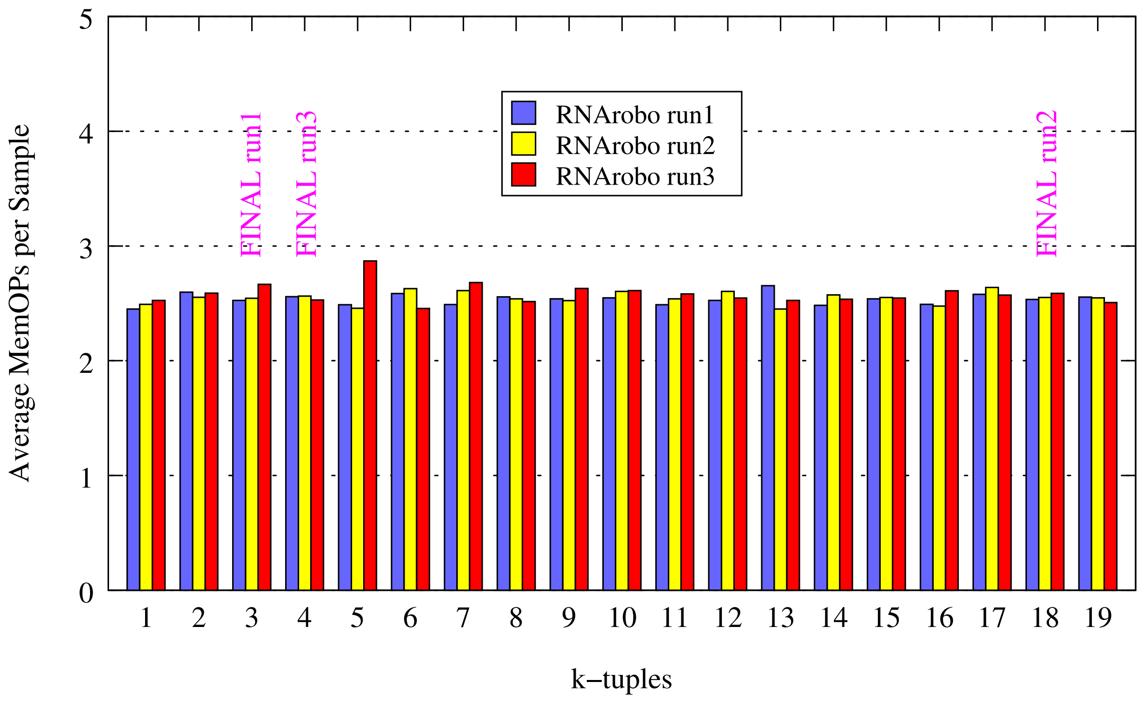


Figure 6.6: **ATP aptamer:** Average number of memory operations for 19 triplets with the best heuristic score (ordered by the heuristic score from left to right). Our heuristic function for this simple motif works well, all the candidates are similar and provide good performance.

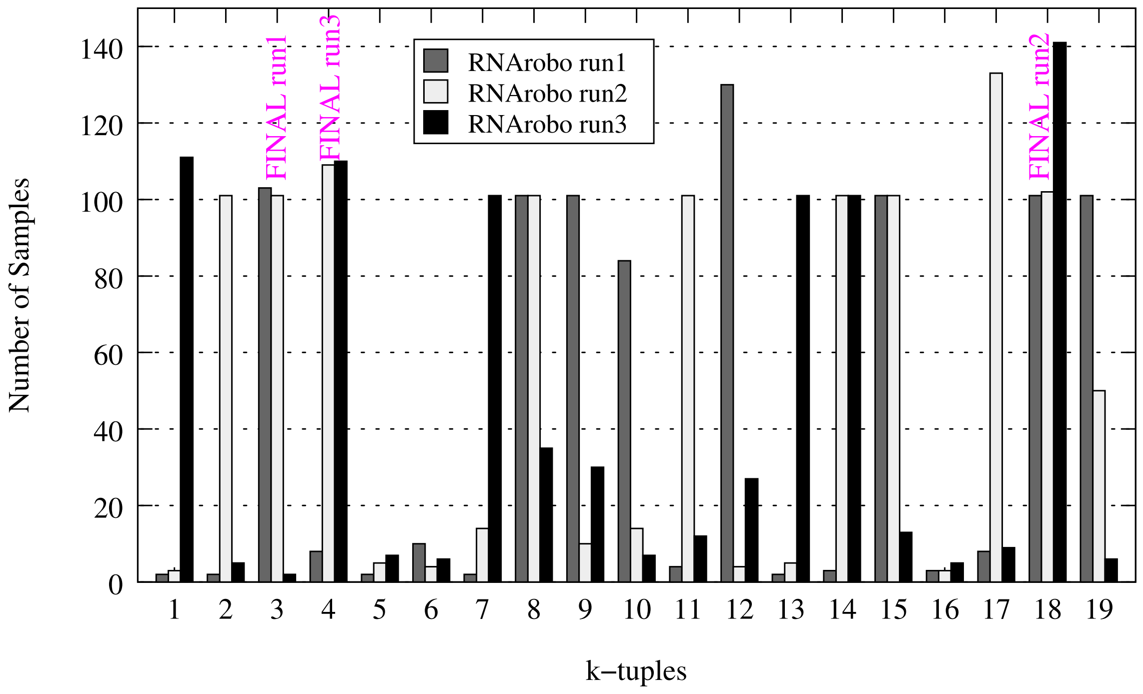


Figure 6.7: **ATP aptamer:** Coverage of the best 19 triplets (ordered by the heuristic score from left to right). The performance of the triplets is similar (Figure 6.6), what causes the higher coverage.

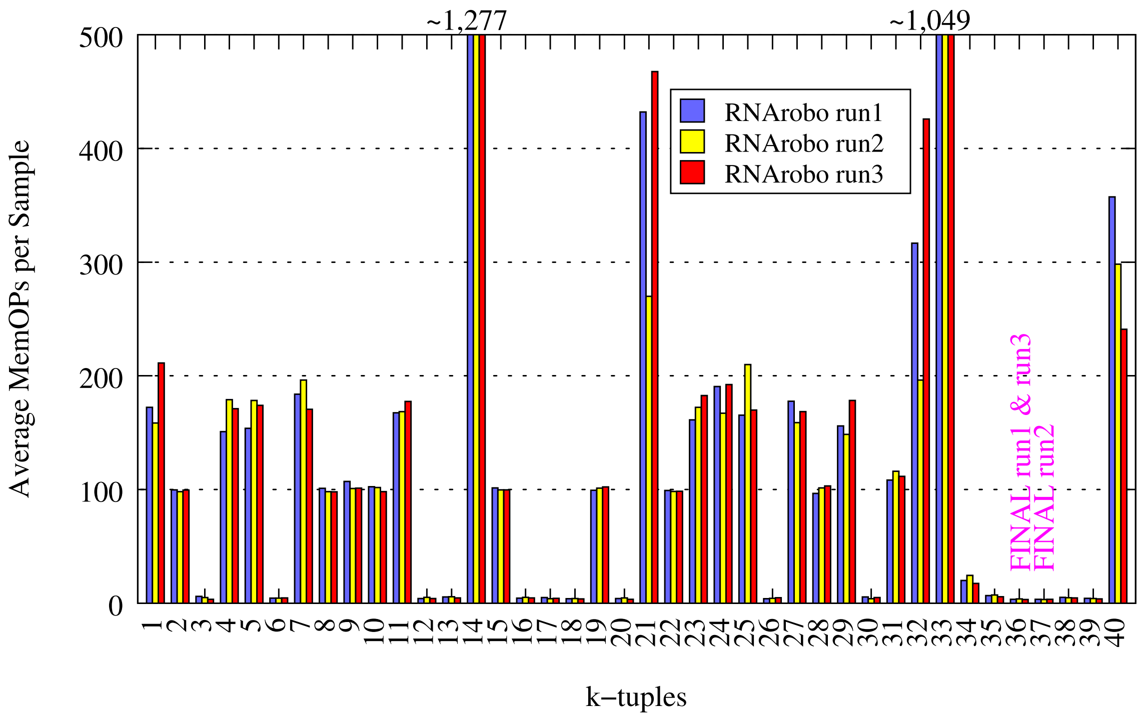


Figure 6.8: **HDV pseudoknot, first triplet:** The average number of memory operations for 40 triplets with the best heuristic score (ordered by the heuristic score from left to right). Many candidates have bad performance, but sufficiently good candidates are proposed as well.

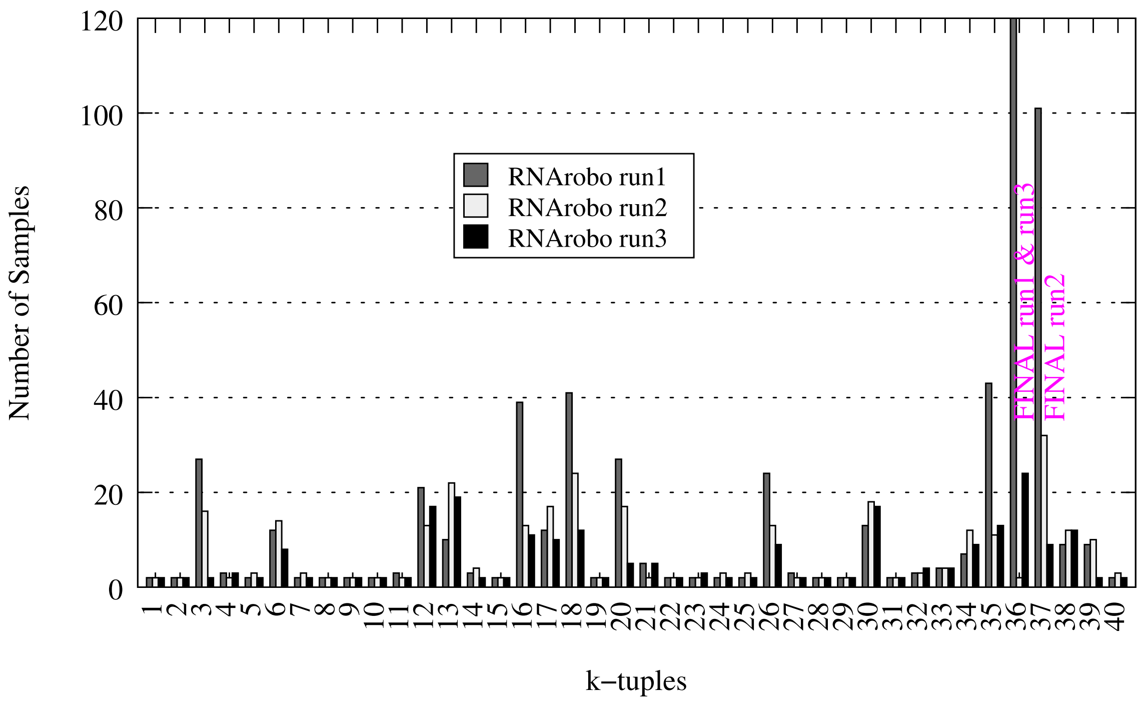


Figure 6.9: **HDV pseudoknot, first triplet:** Coverage of the best 40 triplets (ordered by the heuristic score from left to right). Only good candidates have a high coverage.

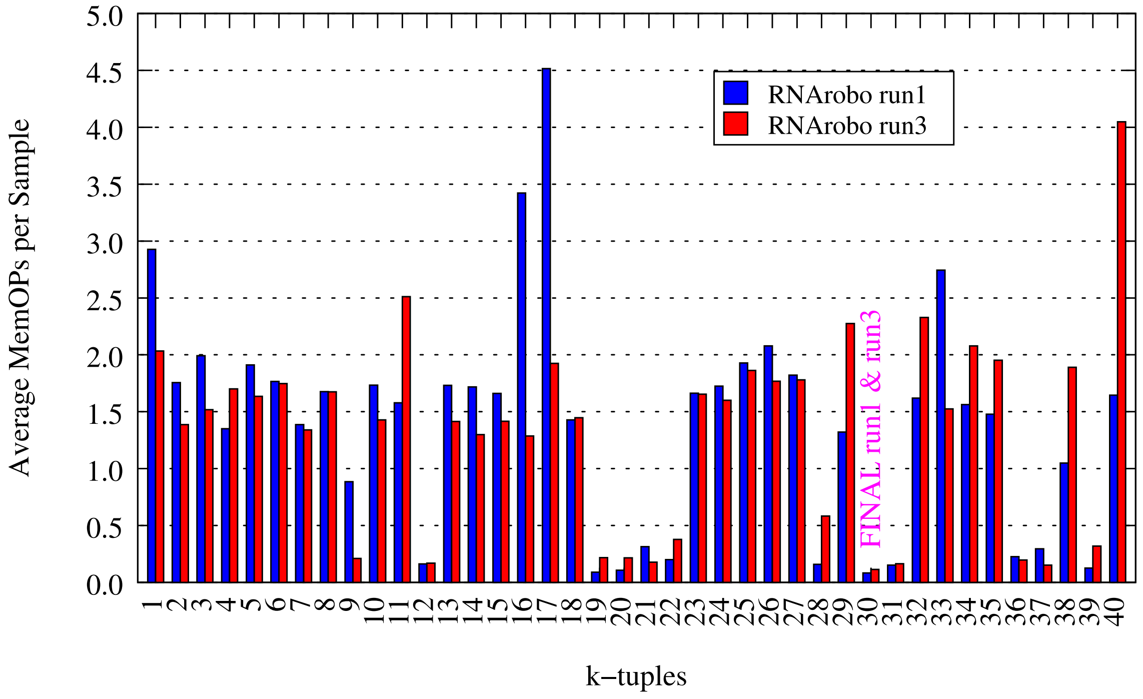


Figure 6.10: **HDV pseudoknot, second triplet in run1 & run3:** The average number of memory operations for 40 triplets with the best heuristic score (ordered by the heuristic score from left to right).

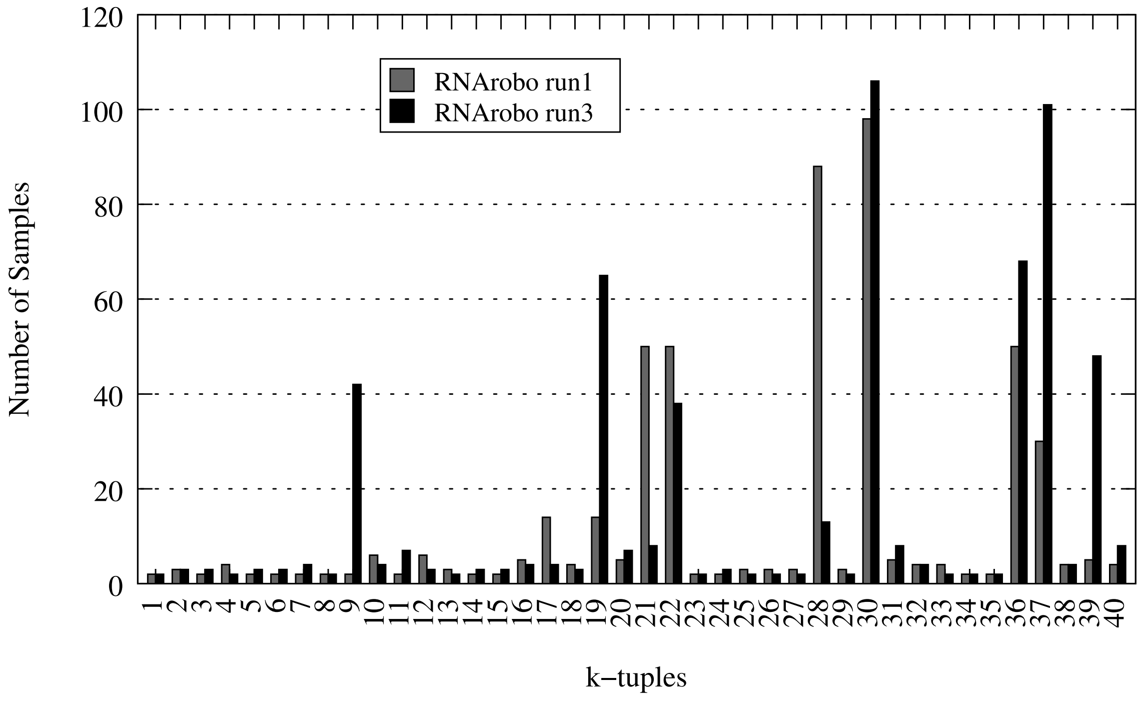


Figure 6.11: **HDV pseudoknot, second triplet in run1 & run3:** Coverage of the best 40 triplets (ordered by the heuristic score from left to right).

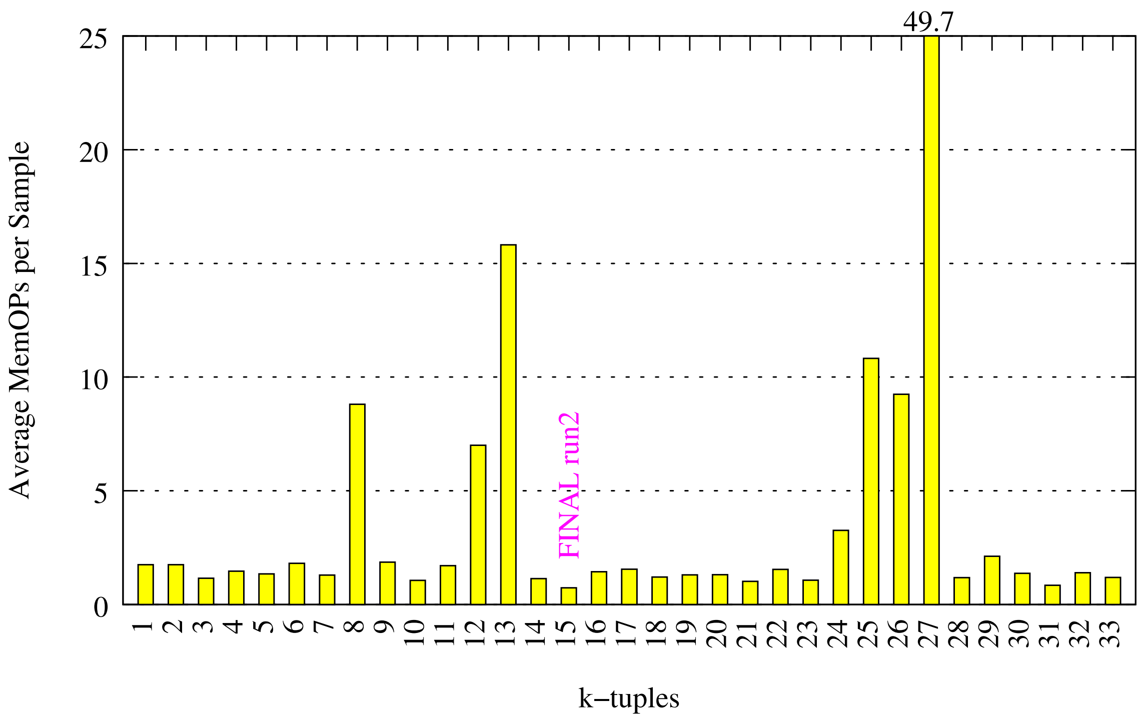


Figure 6.12: **HDV pseudoknot, second triplet in run2:** The average number of memory operations for 33 triplets with the best heuristic score (ordered by the heuristic score from left to right).

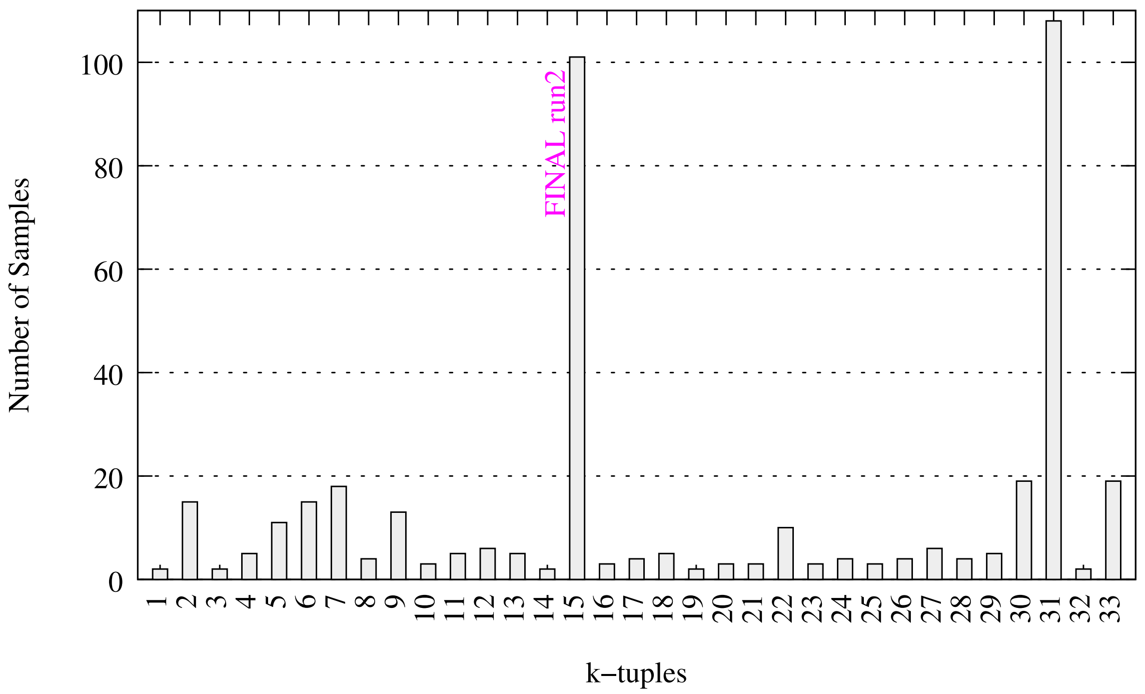


Figure 6.13: **HDV pseudoknot, second triplet in run2:** Coverage of the best 33 triplets (ordered by the heuristic score from left to right).

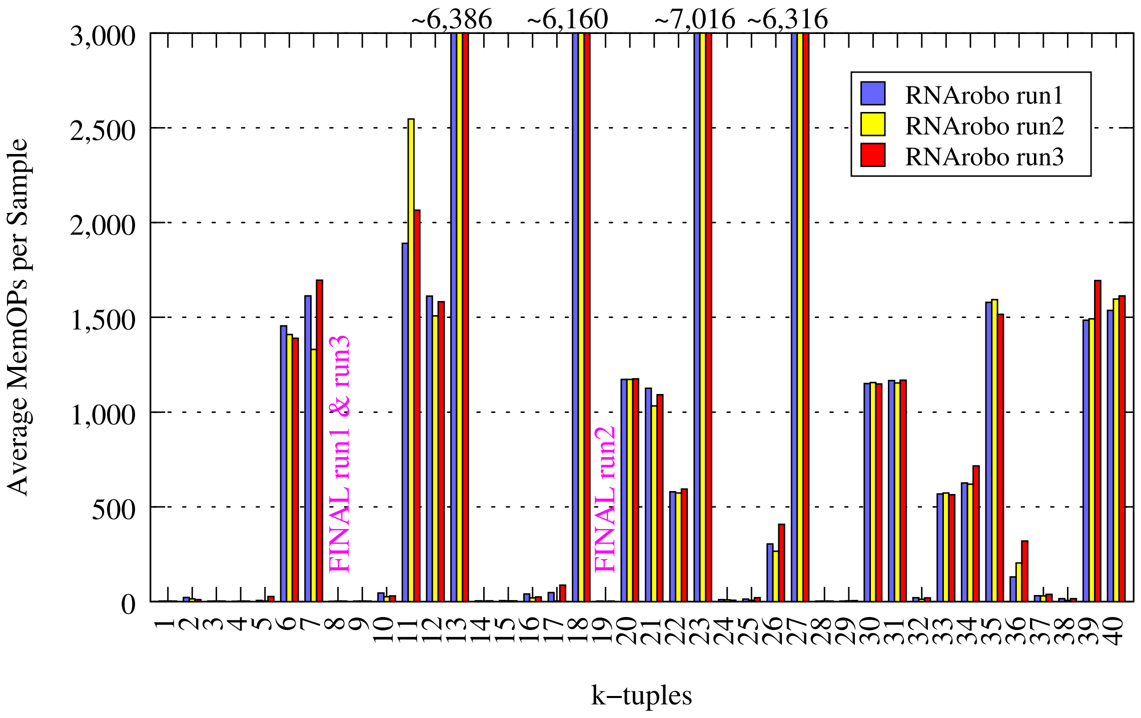


Figure 6.14: **HH ribozyme, first triplet:** The average number of memory operations for 40 triplets with the best heuristic score (ordered by the heuristic score from left to right). Four of the candidates perform really inferior, however there is also many good candidates.

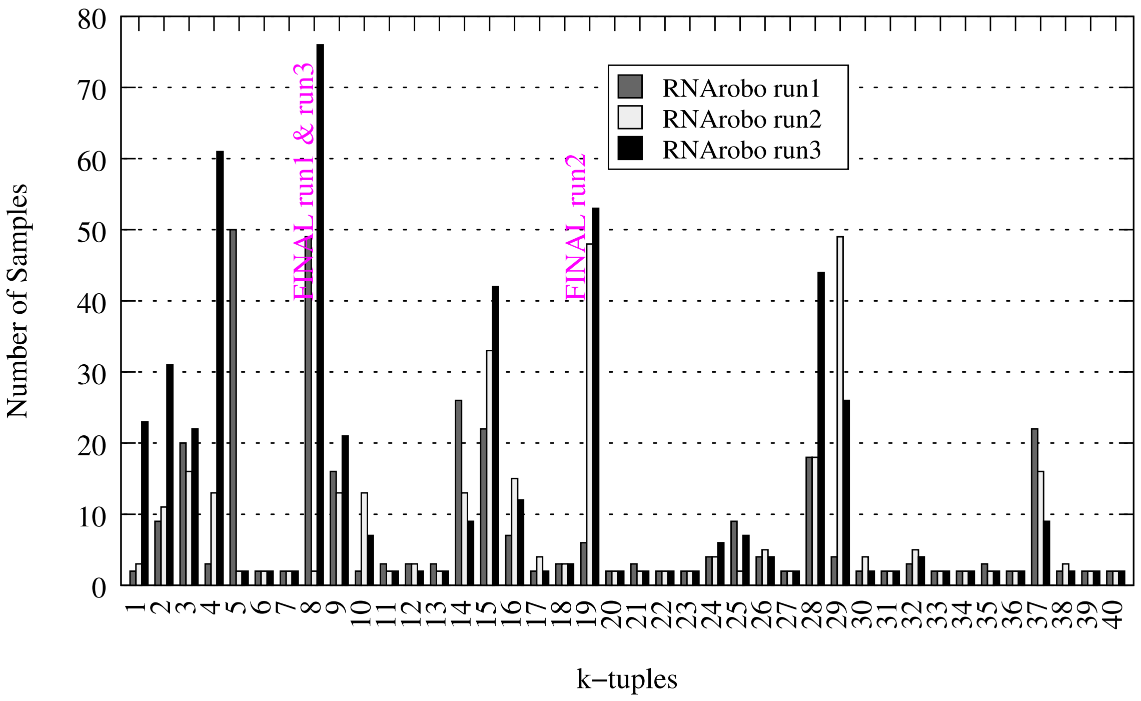


Figure 6.15: **HH ribozyme, first triplet:** Coverage of the best 40 triplets (ordered by the heuristic score from left to right).

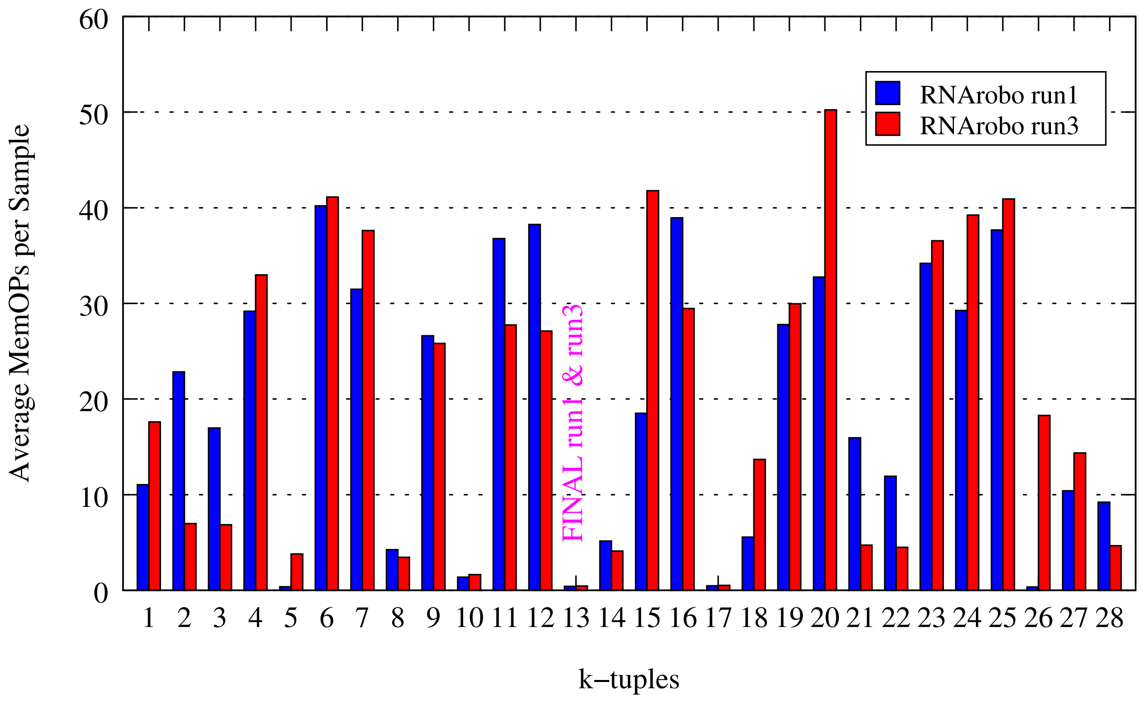


Figure 6.16: **HH ribozyme, second triplet in run1 & run3:** The average number of memory operations for 28 triplets with the best heuristic score (ordered by the heuristic score from left to right).

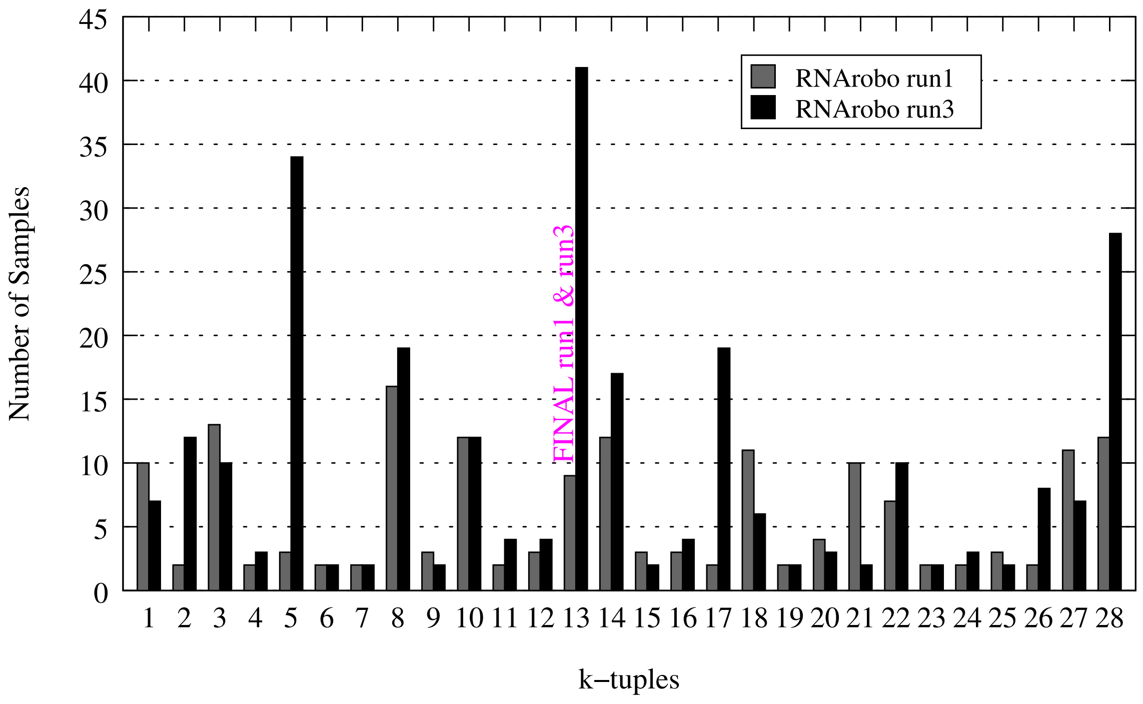


Figure 6.17: **HH ribozyme, second triplet in run1 & run3:** Coverage of the best 28 triplets (ordered by the heuristic score from left to right).

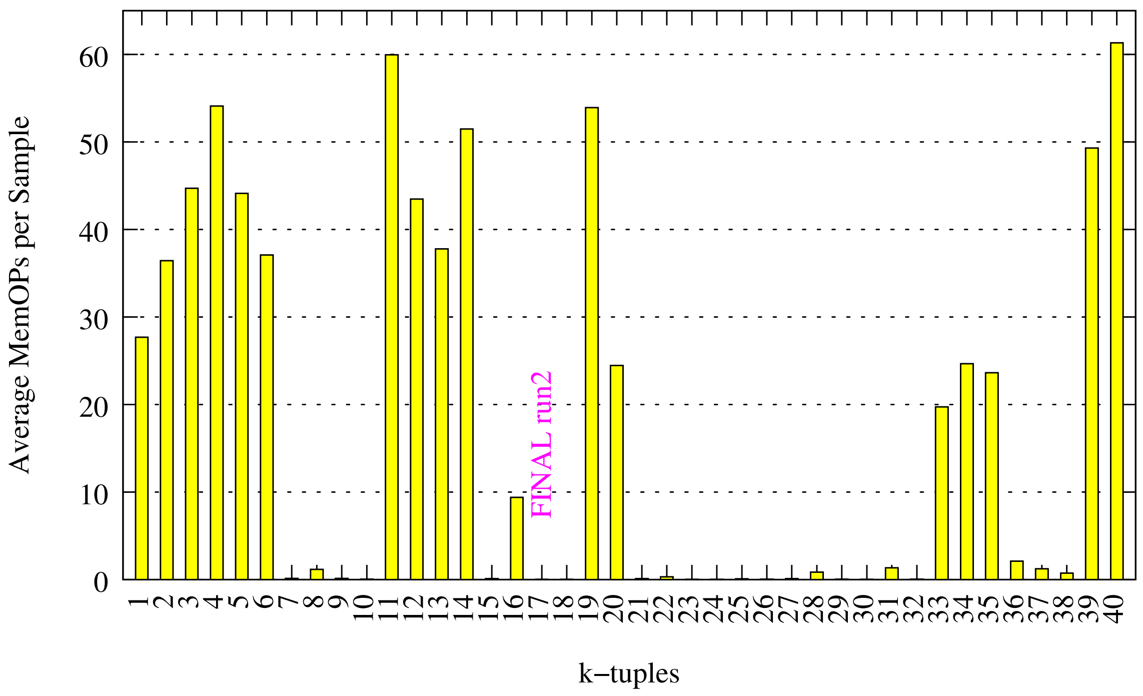


Figure 6.18: **HH ribozyme, second triplet in run2:** The average number of memory operations for 40 triplets with the best heuristic score (ordered by the heuristic score from left to right).

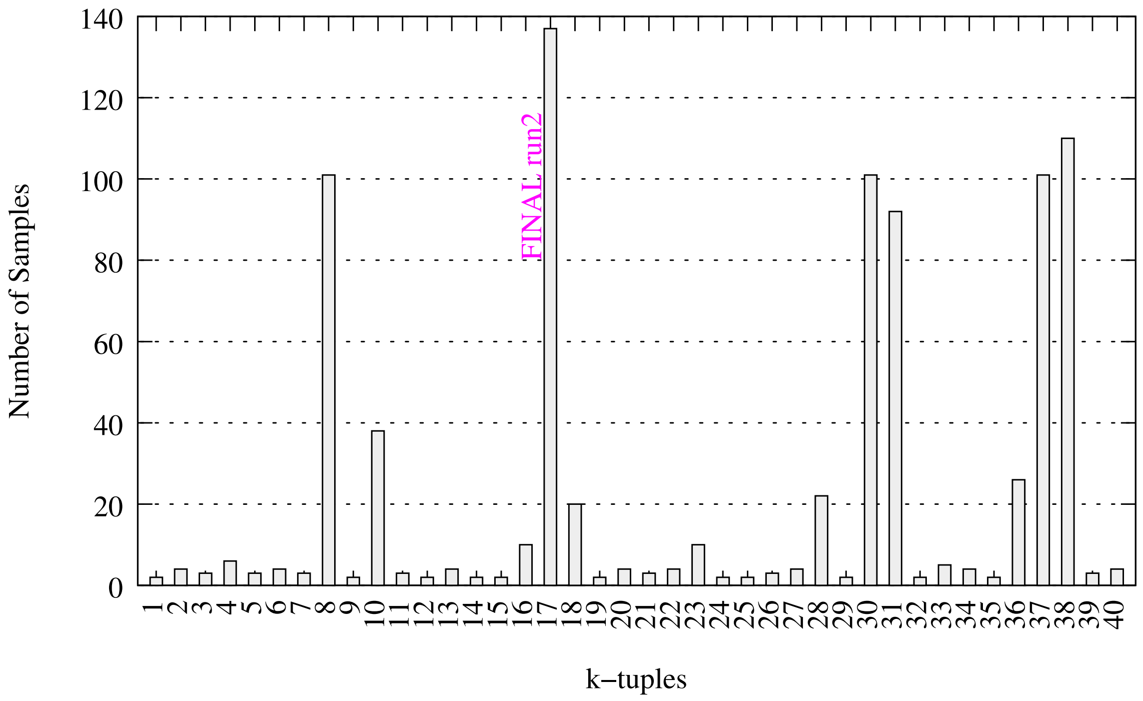


Figure 6.19: **HH ribozyme, second triplet in run2:** Coverage of the best 40 triplets (ordered by the heuristic score from left to right).

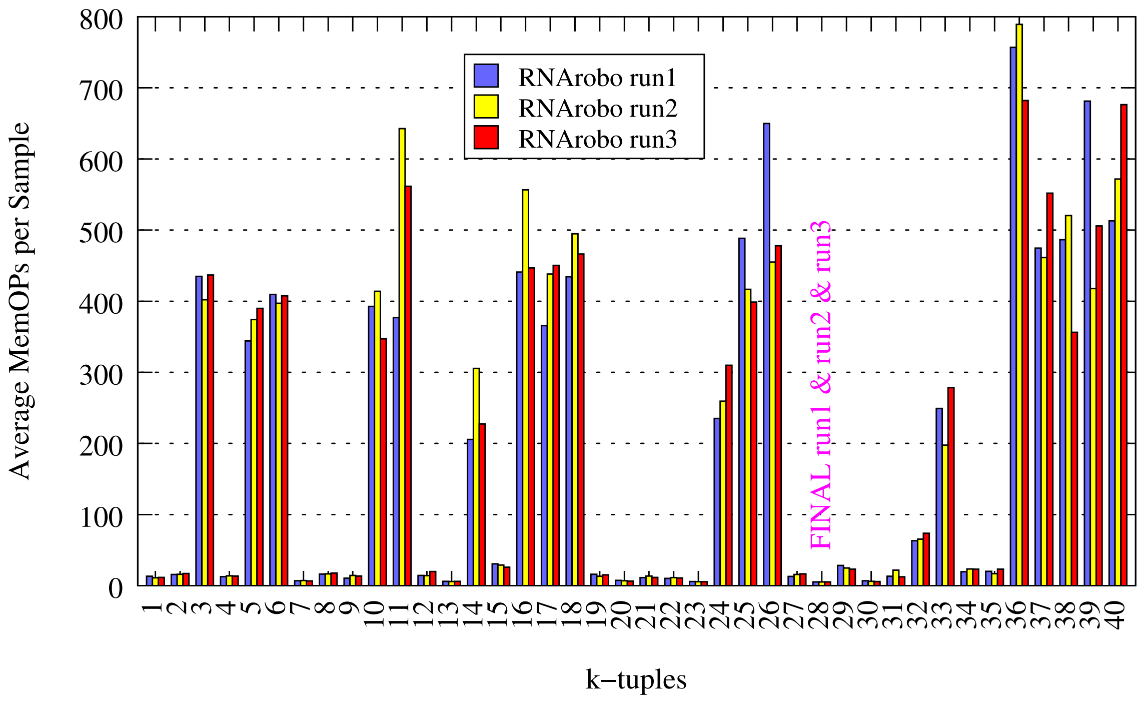


Figure 6.20: **tRNA, first triplet:** The average number of memory operations for 40 triplets with the best heuristic score (ordered by the heuristic score from left to right).

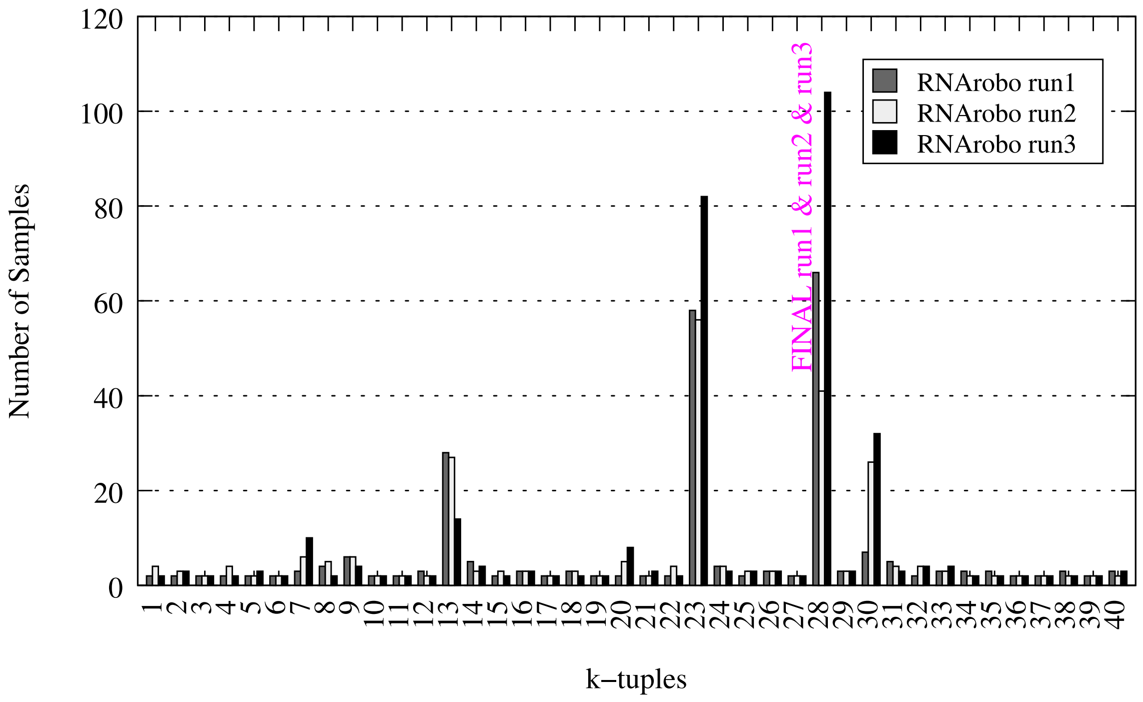


Figure 6.21: **tRNA, first triplet:** Coverage of the best 40 triplets (ordered by the heuristic score from left to right).

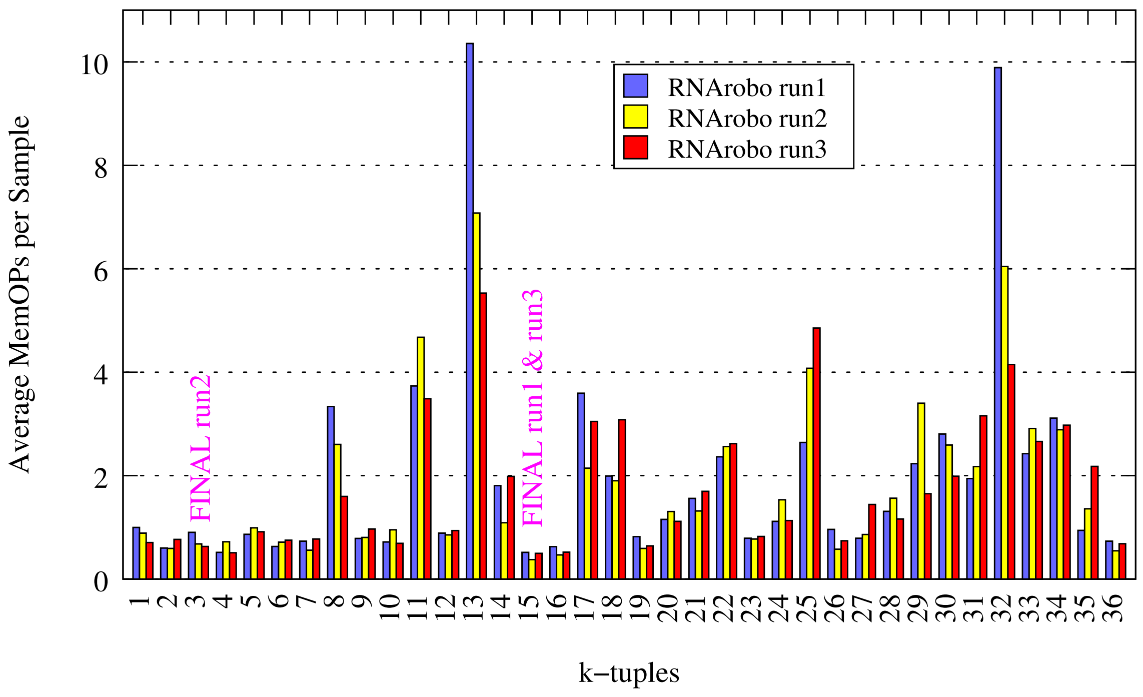


Figure 6.22: **tRNA, second triplet:** The average number of memory operations for 36 triplets with the best heuristic score (ordered by the heuristic score from left to right).

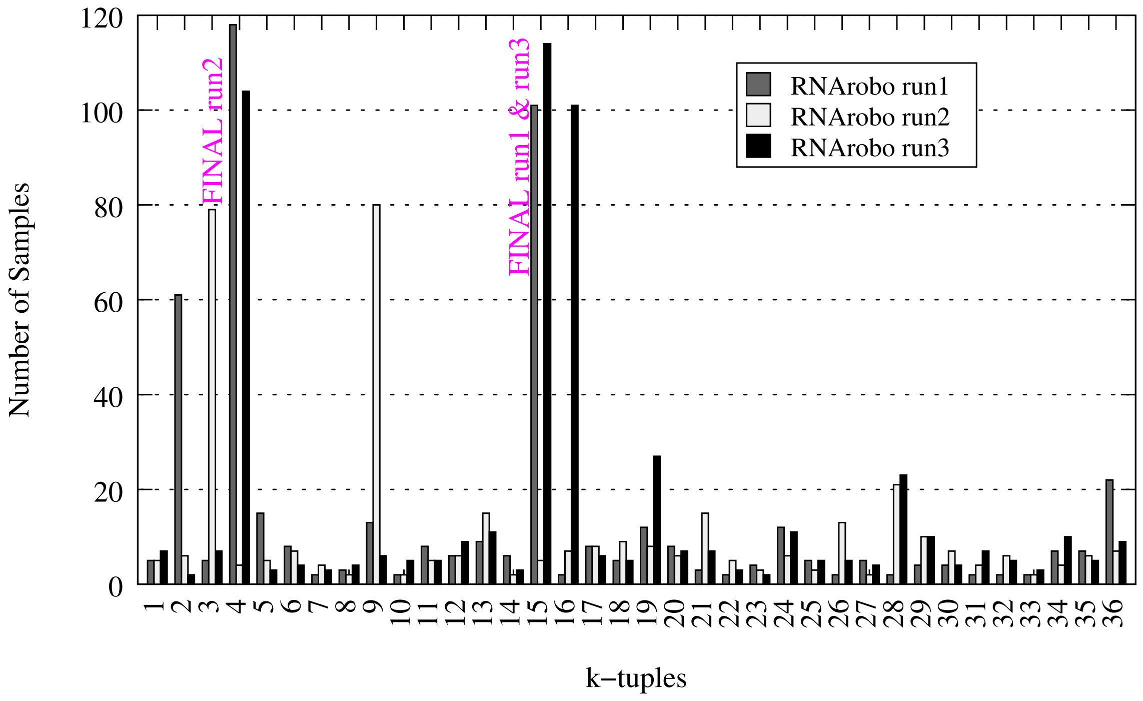


Figure 6.23: **tRNA, second triplet:** Coverage of the best 36 triplets (ordered by the heuristic score from left to right).

Conclusion

In this work we studied the problem of RNA structural motif search from two aspects. Firstly, we studied its computational complexity. Secondly, we enhanced our previous tool [Rampášek, 2010] to be a practical solution for this problem even in large genomic datasets. The main advantage of our tool RNArobo is the ability to search for arbitrarily pseudoknotted motifs with insertions, which we handle in a unique way. In the last part of our work, under the guidance of prof. Andrej Lupták, we developed an *in silico* method for post-processing of the RNArobo search results according to their estimated structural stability.

In the first part of the thesis, we devoted our attention to the study of computational complexity of RNA structural motif search problem (RNA-SMS). We formally defined a generalized problem, the structural motif search problem (SMS), and proved it to be NP-complete. The proof was conducted by a reduction from another NP-complete problem, ONE-IN-THREE 3SAT. We subsequently showed a straightforward modification of the reduction to obtain NP-completeness of RNA-SMS. In our study, we also identified two causes of the NP-completeness: elements of variable length, and pseudoknots in their general form. In further work, we would like to investigate, whether some weaker restrictions on one of these properties would enable us to device an efficient search algorithm. Examples might include limited number of pseudoknots or limits on their relative position (e.g. planar pseudoknots). Restricting pseudoknots in some way in return for efficient algorithm has already been studied for RNA secondary structure prediction problem [Crochemore et al., 2005, Rivas and Eddy, 1999], however not for RNA structural motif search.

Proving RNA-SMS to be NP-complete diminished our hopes for a general solution in deterministic polynomial time. Thus, we have worked to improve our previously proposed RNArobo algorithm to shorten its execution time in practice. RNArobo is a backtracking based algorithm, and as such, it is sensitive to ordering of variables (elements). We proposed a data-driven method for finding a close-to-optimal element ordering. Our method proved to be significantly beneficial in multiple experimental tests. For complex motifs the implementation of RNArobo with this method tends to perform faster than the established tools. This is especially true for pseudoknotted motifs, and for motifs with many allowed distortions, when RNArobo benefits from its robustness. On the other hand, when a motif does not allow for distortions, our dynamic programming is not as efficient as the methods used in the other tools. Thus in future, we would like to further enhance RNArobo time performance. One possible way would be to augment RNArobo by a pre-filtering technique, such that we could quickly filter out large portions of the searched database. This would allow us to focus only on those parts, which exhibit some evidence of a possible motif occurrence.

The third contribution of our work is a set of tools for RNArobo result evaluation. These tools automate the analysis which a biochemist expert would do manually, thus facilitating a high-throughput post-processing of results.

Overall our work resulted in a practical computational pipeline, similar to that proposed in [Jimenez et al., 2012], that can be used to discover new homologs of functional RNAs.

Appendix A

The IUPAC Notation

	Symbol	Meaning	Mnemonic
DNA Bases	A	Adenosine	<u>A</u>denosine
	C	Cytosine	<u>C</u>ytosine
	G	Guanine	<u>G</u>uanine
	T	Thymine	<u>T</u>hymine
Ambiguity Characters	R	G + A	pu<u>R</u>ine
	Y	T + C	p<u>Y</u>rimidine
	S	G + C	<u>S</u>trong interactions (3H bonds)
	W	T + A	<u>W</u>eak interactions (2H bonds)
	K	G + T	<u>K</u>eto
	M	A + C	a<u>M</u>ino
	D	G + T + A	Not-C (<u>D</u> follows C in alphabet)
	H	T + A + C	Not-G (<u>H</u> follows G)
	B	G + T + C	Not-B (<u>B</u> follows A)
	V	G + A + C	Not-T or U (<u>V</u> follows U)
	N	G + A + T + C	a<u>N</u>y

Table A.1: The IPUAC nucleic acid notation

Appendix B

Motif Descriptors

```
# ATP aptamer
#

h1 r2 s1 r3 h4 s2 h4' r3' s3 r2' h1'

h1 0:0 *****:*****
r2 0:0 NNNN:NNNN TGCA
r3 0:0 CNNN:NNNG TGCA
h4 0:0 *****:*****

s1 0 GGAAGAAACTG
s2 0 NNN[17]
s3 0 G
```

Listing B.1: Descriptor for ATP aptamer [Lupták, 2011]

```
# HDV pseudoknot
#

h1 r2 s1 r3 s2 r4 r5 s3 r5' r2' h1' s4 h6 s5 h6' s6 r3' r4'

h1 0:0 R:Y
r2 0:1 NNNNNN:NNNNNN TGCA
r3 0:0 NNNN**:**NNNN TGCA
r4 0:0 NNN:NNN TGCA
r5 0:0 NNN:NNN TGCA
h6 0:1 NNNN****:****NNNN

s1 0   N[14]
s2 0   *
s3 0   TNCNCGY*
s4 0   GN****
s5 0   NNN[20]
s6 0   CNRA*
```

Listing B.2: Descriptor for HDV pseudoknot [Lupták, 2011]

```
# Hammerhead Ribozyme
#

s1 r1 s2 r2 s3 r2' s4 r3 s5 r3' s6 r1' s7

r1 0:0 ***NNN:NNN*** TGCA
r2 0:0 ***NNN:NNN*** TGCA
r3 0:0 ***NNN:NNN*** TGCA

s1 0   NNNNNNNNNNN
s2 0   CTGANGA
s3 0   NNNN[46]
s4 0   GAAA
s5 0   NNNN[46]
s6 0   TN
s7 0   NNNNNNNNNNN
```

Listing B.3: Descriptor for Hammerhead Ribozyme [Lupták, 2011]

```
# tRNA
#

h1 s1 h2 s2 h2' s3 h3 s4 h3' s5 h4 s6 h4' h1' s8

h1 0:2 NNNNNNN:NNNNNNN
h2 0:1 *NNN:NNN*
h3 0:1 NNNNN:NNNNN
h4 0:1 NNNNN:NNNNN
s1 0 TN
s2 0 NNNN**********
s3 0 N
s4 0 NNNNNN*
s5 0 NN*******************
s6 0 TTC****
s8 0 NCCA
```

Listing B.4: Descriptor for tRNA [Eddy, 1996]

Bibliography

[Bellaousov and Mathews, 2010] Bellaousov, S. and Mathews, D. (2010). ProbKnot: Fast prediction of RNA secondary structure including pseudoknots. *RNA*, 16(10):1870–1880.

[Billoud et al., 1996] Billoud, B., Kontic, M., and Viari, A. (1996). Palingol: a declarative programming language to describe nucleic acids' secondary structures and to scan sequence database. *Nucleic Acids Research*, 24(8):1395.

[Chang et al., 2006] Chang, T.-H., Huang, H.-D., Chuang, T.-N., Shien, D.-M., and Horng, J.-T. (2006). RNAMST: efficient and flexible approach for identifying RNA structural homologs. *Nucl. Acids Res.*, 34:W423–428.

[Crochemore et al., 2005] Crochemore, M., Hermelin, D., Landau, G., and Vialette, S. (2005). Approximating the 2-interval pattern problem. *Algorithms–ESA 2005*, pages 426–437.

[Durbin et al., 1998] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press.

[Eddy, 1996] Eddy, S. (1996). RNABob: a program to search for RNA secondary structure motifs in sequence databases. unpublished.

[Gautheret et al., 1990] Gautheret, D., Major, F., and Cedergren, R. (1990). Pattern searching/alignment with RNA primary and secondary structures: an effective descriptor for tRNA. *Comput Appl Biosci*, 6(4):325–31.

[George and Tenenbaum, 2009] George, A. D. and Tenenbaum, S. A. (2009). Informatic resources for identifying and annotating structural RNA motifs. *Mol Biotechnol*, 41(2):180–93.

[Grillo et al., 2003] Grillo, G., Licciulli, F., Liuni, S., Sbisa, E., and Pesole, G. (2003). PatSearch: a program for the detection of patterns and structural motifs in nucleotide sequences. *Nucleic Acids Research*, 31(13):3608.

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

[Jimenez et al., 2012] Jimenez, R., Rampášek, L., Brejová, B., Vinař, T., and Lupták, A. (2012). Discovery of rna motifs using a computational pipeline that allows insertions in paired regions and filtering of candidate sequences. *Methods in molecular biology (Clifton, NJ)*, 848:145.

[Kim and Cohen, 1998] Kim, S. and Cohen, A. (1998). On the Behrens-Fisher problem: A review. *Journal of Educational and Behavioral Statistics*, 23(4):356–377.

[Lathrop, 1994] Lathrop, R. (1994). The protein threading problem with sequence amino acid interaction preferences is NP-complete. *Protein Engineering Design and Selection*, 7(9):1059.

[Lorenz et al., 2011] Lorenz, R., Bernhart, S., zu Siederdissen, C., Tafer, H., Flamm, C., Stadler, P., and Hofacker, I. (2011). ViennaRNA package 2.0. *Algorithms for Molecular Biology*, 6(1):26.

[Lowe and Eddy, 1997] Lowe, T. and Eddy, S. (1997). tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence. *Nucleic acids research*, 25(5):0955–964.

[Lupták, 2011] Lupták, A. (2011). personal communication.

[Lyngsø, 2004] Lyngsø, R. (2004). Complexity of pseudoknot prediction in simple models. *Automata, Languages and Programming*, pages 171–211.

[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. *Nucl. Acids Res.*, 29(22):4724–4735.

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

[NCBI, 2012a] NCBI (2012a). National Center for Biotechnology Information database: genome of Drosophila melanogaster. "`ftp://ftp.ncbi.nlm.nih.gov/genomes/Drosophila_melanogaster/`".

[NCBI, 2012b] NCBI (2012b). National Center for Biotechnology Information database: genome of Homo sapiens. "`ftp://ftp.ncbi.nlm.nih.gov/genomes/H_sapiens/`".

[Nussinov and Jacobson, 1980] Nussinov, R. and Jacobson, A. (1980). Fast algorithm for predicting the secondary structure of single-stranded RNA. *Proceedings of the National Academy of Sciences*, 77(11):6309.

[Rampášek, 2010] Rampášek, L. (2010). RNA motif search in genomic sequences. *Bachelor thesis*, Faculty of Mathematics, Physics and Informatics, Comenius University in Bratislava.

[Rampášek, 2011] Rampášek, L. (2011). RNA structural motif search is NP-complete. *Proceedings of the Student Science Conference 2011*, Faculty of Mathematics, Physics and Informatics, Comenius University in Bratislava.

[Reeder et al., 2007] Reeder, J., Reeder, J., and Giegerich, R. (2007). Locomotif: from graphical motif description to RNA motif search. *Bioinformatics*, 23(13):i392–i400.

[Ren et al., 2005] Ren, J., Rastegari, B., Condon, A., and Hoos, H. (2005). HotKnots: heuristic prediction of RNA secondary structures including pseudoknots. *Rna*, 11(10):1494–1504.

[Rivas and Eddy, 1999] Rivas, E. and Eddy, S. (1999). A dynamic programming algorithm for RNA structure prediction including pseudoknots1. *Journal of molecular biology*, 285(5):2053–2068.

[Ruminski et al., 2010] Ruminski, D., Webb, C., Riccitelli, N., and Luptak, A. (2010). Processing of insect retrotransposons by self-cleaving ribozymes.

[Russell and Norvig, 2010] Russell, S. and Norvig, P. (2010). *Artificial intelligence: A modern approach*. Prentice hall.

[Ruxton, 2006] Ruxton, G. (2006). The unequal variance t-test is an underused alternative to Student's t-test and the Mann–Whitney U test. *Behavioral Ecology*, 17(4):688–690.

[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–i93.

[Satterthwaite, 1946] Satterthwaite, F. (1946). An approximate distribution of estimates of variance components. *Biometrics bulletin*, 2(6):110–114.

[Schaefer, 1978] Schaefer, T. (1978). The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226. ACM.

[Smith, 1936] Smith, H. (1936). The problem of comparing the results of two experiments with unequal errors. *Journal of the Council of Scientific and Industrial Research*, 9:211–212.

[Sperschneider and Datta, 2010] Sperschneider, J. and Datta, A. (2010). DotKnot: pseudoknot prediction using the probability dot plot under a refined energy model. *Nucleic acids research*, 38(7):e103–e103.

[Webb et al., 2009] Webb, C. H., Riccitelli, N. J., Ruminski, D. J., and Luptak, A. (2009). Widespread occurrence of self-cleaving ribozymes. *Science*, 326(5955):953.

[Welch, 1947] Welch, B. (1947). The generalization of 'Student's' problem when several different population variances are involved. *Biometrika*, 34(1/2):28–35.

[Xayaphoummine et al., 2005] Xayaphoummine, A., Bucher, T., and Isambert, H. (2005). Kinefold web server for RNA/DNA folding path and structure prediction including pseudoknots and knots. *Nucleic acids research*, 33(suppl 2):W605–W610.

[Zuker and Sankoff, 1984] Zuker, M. and Sankoff, D. (1984). RNA secondary structures and their prediction. *Bulletin of Mathematical Biology*, 46(4):591–621.