RNA motif search in genomic sequences

BACHELOR THESIS

Ladislav Rampášek

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

9.2.1 Informatics

Supervisor: Mgr. Tomáš Vinař, PhD.

Bratislava 2010

Čestné prehlásenie

Čestne prehlasujem, že moju bakalársku prácu som vypracoval samostatne s pomocou uvedenej literatúry.

V Bratislave 1. júna 2010 Podpis ....

Ladislav Rampášek

Poďakovanie

Ďakujem vedúcemu mojej práce, Mgr. Tomášovi Vinařovi, PhD., a Mgr. Broňi Brejovej, PhD. za podnetné rozhovory a návrhy, ktoré prispeli ku kvalite práce. Ďakujem tiež za poskytnutú technickú podporu.

Ladislav Rampášek

Acknowledgement

I thank my supervisor, Mgr. Tomáš Vinař, PhD., and Mgr. Broňa Brejová, PhD. for many insightful conversations and comments during the development of my thesis. I also thank for provided technical support.

Ladislav Rampášek

Abstrakt

Autor:	Ladislav Rampášek
Názov bakalárskej práce:	Vyhľadávanie RNA motívov v genomických sekvenciách
Škola:	Univerzita Komenského v Bratislave
Fakulta:	Fakulta matematiky, fyziky a informatiky
Katedra:	Katedra informatiky
Vedúci bakalárskej práce:	Mgr. Tomáš Vinař, PhD.
Miesto a rok:	Bratislava, jún 2010

Hlavným cieľom tejto práce bolo vyvinúť nástroj na vyhľadávanie RNA motívov, ktorý by umožňoval vyhľadávať motívy s povolenými chybami ako sú inzercie a delécie popri chybách ako nezhoda v primárnej sekvencií a chyba komplementarity v sekundárnej sekvencií, ktoré umožňujú súčasné nástroje (napr. RNABob). Táto práca bola motivovaná praktickou potrebou biochemika M.S. Andreja Luptáka PhD. z University of California at Irvine, ktorý používa RNA vyhľadávacie nástroje pri hľadaní nových RNA génov so špecifickou funkcionalitou.

Na začiatku práce sa venujeme významu a štruktúre RNA. Uvádzame viacero spôsobov ako sa dá definovať RNA motív pre účely jeho vyhľadávania.

Pre zvládnutie problematiky inzercií a delécií pri vyhľadávaní RNA motívov sme vyvinuli nový nástroj RNAMot2, ktorému sa venujeme vo väčšine tejto práce. Algoritmus implementovaný v RNAMot2 pozostáva z dvoch základných častí. Prvou časťou je algoritmus na vyhľadávanie výskytov štrukturálnych elementov daného RNA motívu, ktorý je založený na dynamickom programovaní. Druhou časťou je backtracking algoritmus použitý v už existujúcom nástroji RNAMot, ktorý slúži na skladanie jednotlivých výskytov elementov do výskytu celého motívu. Formát deskriptoru pre náš RNAMot2 je odvodený od formátu používaného programom RNABob, ktorý sme doplnili o možnosť povoliť inzercie v sekvenciách štrukturálnych elementov daného motívu.

Nový nástroj sme empiricky overili v testoch, kde sa potvrdilo, že vďaka možnosti povoliť inzercie a delécie náš nástroj dokáže nájsť viac biologicky hodnoverných výskytov hľadaného motívu ako súčasné nástroje. Hoci sa ukázalo, že RNAMot2 výrazne zaostáva v rýchlosti vyhľadávania, uvádzame záverom práce viacero možných vylepšení od algoritmických po implementačné, ktorých zapracovanie do RNAMot2 by prinieslo značné zlepšenie výkonu.

Abstract

Author:	Ladislav Rampášek
Title:	RNA motif search in genomic sequences
University:	Comenius University in Bratislava
Faculty:	Faculty of Mathematics, Physics and Informatics
Department:	Department of Computer Science
Supervisor:	Mgr. Tomáš Vinař, PhD.
Place and year:	Bratislava, June 2010

The ultimate goal of this work was to develop an RNA motif search tool that, besides defects allowed by current tools (e.g. RNABob), also allows insertions and deletions in the occurrences of the motif elements. The work was motivated by practical needs of a biochemist Dr. Andrej Lupták from University of California at Irvine, who uses RNA motif search tools to discover new RNA gens of specific functions.

Early in the thesis we take a more detailed look on the structure of RNA motifs and explain their role and importance. We also show how descriptors are used to define permissible deviations from the prescribed motif.

To address the problem of insertions and deletions in RNA motif search, we have developed a new search tool RNAMot2 and we dedicate most of the text to its description and evaluation. The algorithm implemented in RNAMot2 consists of two parts: the new algorithm for search of individual motif elements based on dynamic programing, and the backtracking algorithm used in an existing tool RNAMot. RNAMot2 uses motif descriptors compatible with RNABob, that we have extended to handle insertions.

Our empirical results have shown that RNAMot2 can be more sensitive than current tools due to the ability to allow insertions and deletions, however is much slower. Nevertheless, in the end of the thesis we propose several ways how to significantly enhance the RNAMot2 time performance in the future.

Contents

Introduction **1**

1 Biological Background **2**

1.1 Functions and Types of RNA . . . 2
1.2 RNA Structure . . . 2
1.3 The Task . . . 4
1.4 RNA Motif Descriptors . . . 5
1.4.1 RNAMot Descriptor Format . . . 5
1.4.2 RNABob Descriptor Format . . . 6
1.4.3 RNAMotif Descriptor Format . . . 7

2 Existing RNA Motif Search Tools **8**

2.1 The Backtracking Method . . . 8
2.1.1 RNAMot . . . 9
2.1.2 RNAMotif . . . 11
2.2 Automaton Approach . . . 12
2.2.1 RNABob . . . 12

3 The New Algorithm **13**

3.1 Dynamic Programming Used in RNAMot2 . . . 13
3.1.1 The Recurrence for Single Strand Elements . . . 13
3.1.2 The Recurrence for Helical Elements . . . 14
3.2 Extension of RNABob Descriptor . . . 15
3.3 Implementation Notes . . . 16
3.3.1 Search Domain Computation . . . 16
3.3.2 The Search for Structural Elements . . . 17
3.4 Input and Output Formats . . . 18

4 Experimental Results **19**

4.1 Motifs Used in the Evaluation . . . 19
4.2 Used Genomic Sequences . . . 19
4.3 Testing Environment . . . 20
4.4 Methodology . . . 20
4.5 The Results . . . 20
4.5.1 R17 . . . 21
4.5.2 Bulge . . . 22
4.5.3 Pseudoknot . . . 22
4.5.4 tRNA . . . 22
4.5.5 HDV ribozyme . . . 22
4.5.6 Sassanfar aptamer . . . 23
4.6 Results Summary . . . 23

5 Future Work **28**
5.1 Suitable Search Order 28
5.2 Filtering 28
5.3 Faster Data Structure 29
5.4 Trade Memory for Speed 29
5.5 Parallelism 29

Conclusion **30**

A Appendix: The IUPAC Notation **31**

B Appendix: Used Descriptors **32**
B.1 R17 32
B.2 Bulge 32
B.3 Pseudoknot 33
B.4 tRNA 34
B.5 HDV ribozyme 36
B.6 Sassanfar aptamer (adenosin) 38

References **40**

Introduction

In this thesis we are going to deal with a bioinformatics problem. In particular, our problem is directly motivated in biology and we need to use computer science and its instruments to solve it. From this point of view we can understand the RNA motif search problem as a problem of searching a text pattern with some additional constraints that are not present in classic text searching problems. These constraints make the search considerably more difficult.

In the process of translation from sometimes loose biological motif definitions to a formal language of computer science, we often have to sacrifice some biological information in return for efficient algorithms. Today we know that spatial structure that a motif adopts in 3D is an important piece of information and cannot be ignored. However, searching with secondary structure (2D) restrictions seems to be a good trade, as 3D constraints are too demanding. Also Webb et al. [2009] in their paper claim that even though sequence-based search can uncover conserved regions, a structural-based approach has proven to be more effective at finding new functional RNAs. However, many current RNA motif searching tools are not capable to handle even simple mutations such as insertions and deletions, which may occur. This results in loosing sensitivity.

Our goal is to come up with a solution that is capable of handling insertions and deletions, which means increase of search sensitivity. Simultaneously, we want the search to run sufficiently fast in practice. However, also after adding the possibility of insertions and deletions, it is still art of its own, to express all the essential specific information defining a motif. We leave this job of revealing which parts and substructures of an RNA motif are the 'carriers' of its functionality, to the biology experts.

1 Biological Background

As we are going to search for RNA motifs, we should have a basic understanding of RNA functionality and structure. To this goal we dedicate this section. We also define the RNA motif search problem later in the section. The majority of knowledge we present here on RNA, we have acquired from Durbin et al. [1998], Brazma et al. [2001].

1.1 Functions and Types of RNA

The first widely known RNA function was its role of a passive intermediary messenger in translation machinery of genes and proteins from DNA. This type of RNA is called the messenger RNA (mRNA), and carries information from DNA to the ribosome, where protein synthesis (translation) takes place. The coding sequence of the mRNA determines the amino acid sequence of the produced protein.

However, the research over past decades shows that there exist many RNAs that do not code any protein. These types of RNA are called non-coding RNAs (ncRNA). Non-coding RNAs adopt sophisticated three-dimensional structures, and some even catalyze biochemical reactions. The best-known examples of ncRNAs are transfer RNAs (tRNAs) and ribosomal RNAs (rRNAs), both of which are involved in the process of translation. There are also non-coding RNAs involved in gene regulation, RNA processing, and other roles. Certain RNAs are able to catalyse chemical reactions such as cutting and ligating other RNA molecules. Since the startling discovery of catalytic RNAs in the early 1980s, a number of new structural and catalytic RNAs have been discovered.

1.2 RNA Structure

Ribonucleic acid (RNA) is a polymer, like DNA, constructed from nucleotides. 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, what stand for <u>A</u>denine, <u>C</u>ytosine, <u>G</u>uanine and <u>U</u>racil. In contrast, DNA uses thymine (T) instead of uracil. Thymine is not found in RNA. Because of this minor difference RNA does not form a double helix, 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. 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, also non-canonical pairs 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. Other pairs form as well. Non-canonical pairs distort regular A-form RNA helices. These distortions, being a reason for RNA spatial structure deviation, seem to be crucial for functionality of some non-coding RNAs. This also suggests that for some types of RNA, the spatial structure is much more important than the sequence itself.

As stated above, RNA, unlike DNA, is typically produced as a single stranded molecule

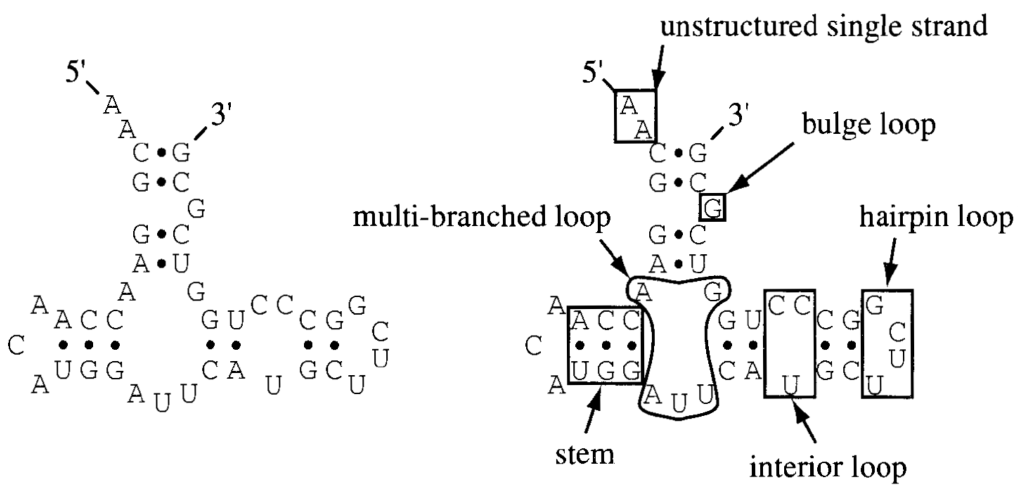


Figure 1: An example of an RNA secondary structure. [Durbin et al., 1998]

which then folds intramolecularly to form a number of short base-paired stems (or also called as helices). This base-paired structure is called the secondary structure of the RNA, and the form which it adopts in 3D space is called the ternary structure. We distinguish the following secondary structure elements:

Stem (helix) is composed of two subsequences whose every nucleotide base forms a pair with the opposite base of the second subsequence. These subsequences are called *strands* of the helix (stem).

Loop is a structure consisting of single-stranded subsequences connected together by base pairs.

Hairpin loop is a loop at the end of a helix.

Hairpin is a simple substructure consisting of a simple stem and loop (it is called so because the structure resembles a hairpin when drawn).

Bulge. Single stranded bases occurring within a stem are called a bulge if the single stranded bases are on only one side of the stem.

Interior loop. Similarly to a bulge, an interior loop is formed of single stranded bases occurring within a stem. Unlike a bulge, there must be single stranded bases interrupting both sides of the stem.

Multi-branched loop is a loop from which three or more stems radiate.

Pseudoknot is a structure composed of at least two helices that disturb a nested fashion of base pairs in RNA secondary structure. That is when a strand of one helix lies in between strands of another helix.

Examples of secondary structure elements can be seen in Figure 1, for a pseudoknot please refer to Figure 2.

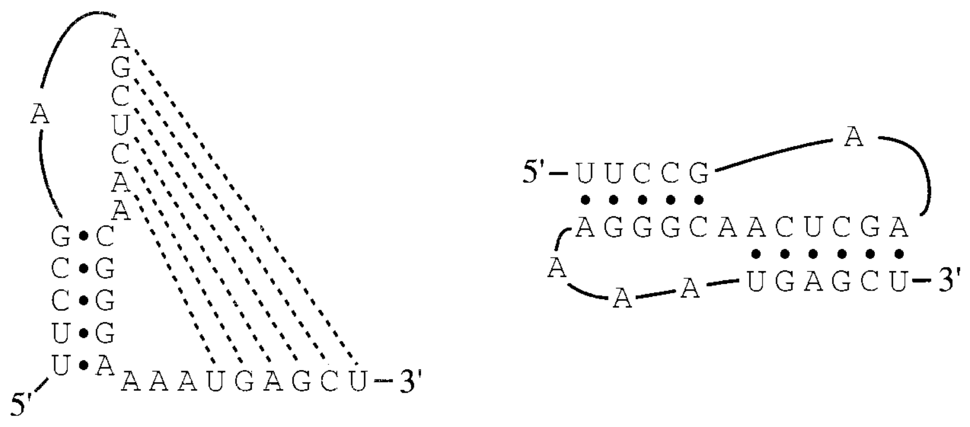


Figure 2: Two different representations of a pseudoknotted structure. The representation in the left emphasizes which base interactions (dotted) cause the pseudoknot. [Durbin et al., 1998]

More formally, we can define a secondary structure of an RNA strand of length n indexed $1 \dots n$ (starting at the 5' end) as a set S of pairing interactions (i_k, j_k) between i_k-th and j_k-th base of the strand,

$$S = \{(i_1, j_1), (i_2, j_2), \dots, (i_m, j_m) \mid \forall k : 1 \le i_k < j_k \le n;\ \nexists k \ne l : i_l = i_k \vee j_l = j_k \vee i_l = j_k\}$$

The condition $\nexists k \ne l : i_l = i_k \vee j_l = j_k \vee i_l = j_k$ says that every base may form up to one pair.

By this definition, we can distinguish two fundamental elements of RNA secondary structure – single-stranded regions and helical regions.

If we add one more condition that S does not contain any two base pairs (i, j) and (i', j') such that $i < i' < j < j'$, then we call such a structure pseudoknot-free, because the condition forbids pseudoknots. The language of pseudoknot-free RNA secondary structures is context-free, however the crossing interactions of pseudoknots in full generality would require context-sensitive grammars [Durbin et al., 1998]. Therefore for many purposes, as for example database searching for RNA homologues or RNA secondary structure prediction, it is usually acceptable to sacrifice the information in pseudoknots in return for efficient algorithms.

1.3 The Task

It is relatively common to find occurrences of homologous RNAs that have a common secondary structure, although their sequences are not so significantly similar. Drastic changes in sequence can often be tolerated as long as compensatory mutations maintain base-pairing complementarity. Thus, it would be advantageous to be able to search for conserved secondary structure in addition to conserved sequence when searching databases for homologous RNAs.

Consider the following definition of the RNA secondary structure motif search problem:

Input: – An RNA secondary structure motif with primary structure restrictions and permissible defects definition.

– A genomic sequence over the alphabet $\{A, C, G, T, U, N\}$, where 'N' stands for any character and 'U' is treated as 'T'[1].

Task: Find all occurrences of the motif in the sequences. A motif match occurrence must meet the motif constraints with up to admissible amount of defects.

Output: For every place where a match occurs, return its position in the sequence and the corresponding matching subsequence.

Note that majority of current tools enable to permit only defects of two types: mismatches in primary sequence and mispairs in secondary structure. A mispair occurs when matched bases in positions which should have been paired are not complementary.

Our goal is to develop a tool, which enables also insertions and deletions of a nucleotide in the pattern of the structure element. The deletion is equivalent to "not inserting" this nucleotide in the sequence that already does not contain this nucleotide. Since, deletions can be reformulated as insertions, the problem reduces to insertions.

1.4 RNA Motif Descriptors

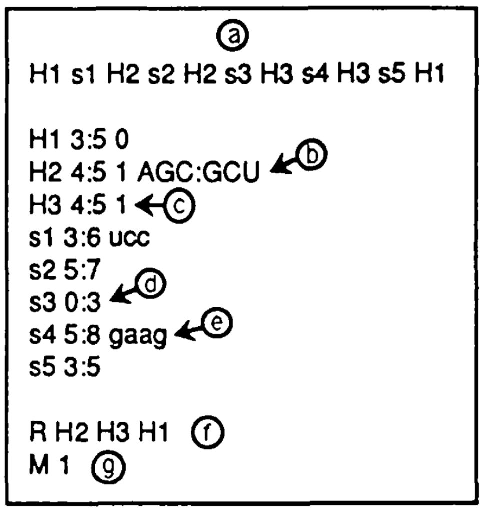
Here we present formats used to describe an RNA secondary structure motif. These formats, called descriptors, are formats used by existing tools with which we deal in the following section. A descriptor is composed of both higher-order structural elements and simple sequence patterns. Descriptors of RNAMot [Gautheret et al., 1990] and particularly RNABob citepeddy1996 are important, as we use it in our new tool.

1.4.1 RNAMot Descriptor Format

The RNAMot descriptor format is a list of structural elements. The list establishes elements position in the motif and is followed by further specification of every element. RNAMot distinguishes two types of structural elements: single-stranded regions (noted as 's') and helical regions (noted as 'H'). Helix notations are present twice in the list to define the positions of both strands composing the helix. Every helix strand is than described by a single-stranded motif. Thus, any arrangement of structural elements in the list is permitted, including pseudoknots. The definition of 's' regions does not exclude the possibility that contained bases may be paired, however such hypothetical pairs are not considered in the search.

More exactly, the list is called the 'map' of structural elements (see Figure 3a) and is followed by a description of the properties of each structural element. For helices ('H') are defined properties in this order: the minimal length, the maximal length, the number of mismatches permitted and finally any primary sequence restrictions (Figure 3b and c). Then for single-stranded elements ('s') the same properties (except mismatches) can be used (Figure 3d and e).

[1]for the purpose of the RNA motif search there is no difference between uracil and thymine



```
H1 s1 H2 s2 H2 s3 H3 s4 H3 s5 H1

H1 3:5 0
H2 4:5 1 AGC:GCU
H3 4:5 1
s1 3:6 ucc
s2 5:7
s3 0:3
s4 5:8 gaag
s5 3:5

R H2 H3 H1
M 1
```

Figure 3: An example of RNAMot descriptor. [Gautheret et al., 1990]

Primary sequence constraints (patterns) are described by using the IUPAC notation[2]. The last two lines of the descriptor are optional declarations of the order in which elements are searched (Figure 3f, we describe the role of the search order in the next section), and the total number of base mismatches allowed for the motif (Figure 3g).

In RNAMot descriptors bases that must be present in a motif match are indicated by upper-case characters and those optional bases by lower-case characters.

1.4.2 RNABob Descriptor Format

Both RNABob algorithm and descriptor format are derived from RNAMot. However, RNABob brings many enhancements, but also leaves out some features. In RNABob descriptor there is no difference in meaning of upper-cased and lower-cased nucleotide bases, also neither the element search order nor the total number of allowed base mismatches can be specified. But the idea of specifying motif topology (map) first, followed by exact description of elements properties, remains the same. However, the way of specifying properties of an element is rather new.

Single strand elements have three attributes: the name of the element in the topology map, the number of allowed mismatches in this element, and the primary sequence constraints that apply to this element.

Properties of a helix are specified in a similar manner, so long as mismatches and primary sequence constraints are given as pairs, separated by a colon ":". For primary sequence constraints, the left side is the constraint applied to the upper strand and the right side is applied to the lower strand of the helix. For mismatches, the left side constraints the number of mismatches in the upper strand, while the number on the right constrains the number of mispaired positions in the helix.

The primary sequence constraint can be given as a sequence of characters according to the IUPAC notation. Allowed length variations are specified with asterisks $*$, where each $*$ allows either 0 or 1 'N' at that position. Note that if the number and types (gap or

[2]see Appendix A

```
h1 r1 h2 s1 h2' r1' h1'

h1 0:0 NNN:NNN
r1 0:0 R:N GNAN
h2 0:0 **NC:GN**
s1 0   UUCG
```

Figure 4: An example of RNABob descriptor.

identity) of positions do not match on the left and the right side of a helix, the descriptor is invalid.

RNABob also introduces a new type of elements – relational elements. Relational element is a generalization of a helical element. In the descriptor, it is marked as 'r'. A relational element has in comparison with a helical element one more property called *transformation matrix* of four nucleotides. The transformation matrix specify the rule for how complementary bases to A-C-G-T are determined in the strands of the relational element. Helical elements are then nothing more then relational elements with implicitly defined transformation matrix as TGYR (RNABob allows G-U pairings for helices by default).

1.4.3 RNAMotif Descriptor Format

The motif description for RNAMotif is complex and consists of four sections called *parameters*, *descriptor*, *sites* and *score*. In the section called *parameters* are defined variables used in the rest of the motif description. The *descriptor* section defines the criteria required to generate a match. Relations among the elements are defined in the section *sites*, while the *score* section ranks found matches to the constraints defined by the user. RNAMotif descriptor format rather reminds a scripting language, specially the *score* section. For more details see the paper by Macke et al. [2001] and RNAMotif user's manual.

We have been working with fairly simple RNAMotifs, as we do not need the *sites* and *score* sections for our purposes. Examples of such RNAMotif descriptor files can be seen in Appendix B.

2 Existing RNA Motif Search Tools

Since early 90's several publicly available RNA motif search tools have been developed starting by RNAMot [Gautheret et al., 1990]. In the paper by Gautheret et al. [1990], the authors define the RNA search problem and devise a structural descriptor for RNA secondary structure and an algorithm based on backtracking. More efficient implementation of RNAMot appeared in 1996 under the name of RNABob by Eddy [1996]. The underlying algorithm uses a nondeterministic finite state automaton with node rewriting rules. Unfortunately, the details of the algorithm has never been documented. Later, numerous stand-alone and web-service based tools, such as Palingol [Billoud et al., 1996], RNAMotif [Macke et al., 2001], PatSearch [Grillo et al., 2003], RNAMST [Chang et al., 2006], and more, were developed allowing users to define RNA motifs according to their requirements and search for these motifs in sequence databases. RNABob, RNAMotif and particularly RNAMot are important for this thesis and therefore will be discussed later in this section in more detail.

Besides these general purpose motif search tools, tools optimized for search of specific structural RNA molecules were devised, e.g. tRNAscan-SE [Schattner et al., 2005] to identify transfer RNA in an RNA sequence, and Riboswitch finder [Bengert and Dandekar, 2004] for riboswitch RNA. For extensive review of current tools, see George and Tenenbaum [2009].

In this work, we are interested in general tools allowing for search of arbitrary RNA motifs. These tools can be basically divided into two categories, according to how they work with a search database: those working on-line and those that need preprocessing. Further, we can discriminate on the basis of the paradigm used for the search. According to this, we distinguish tools based on backtracking, and tools using automaton approach.

2.1 The Backtracking Method

The central idea of the backtracking is dividing a motif into several elements (such as helical regions and single-stranded regions). These elements are subsequently organized to build a search tree. The search tree is than tried to be matched with the searched sequence element by element, progressing from leafs up to the root. Algorithms mainly differ in how the tree is built and matched to a sequence.

The advantage of this approach is the possibility of searching motif elements in arbitrary order. When this order is chosen well, that is when we choose motifs of high specificity first, the search time can be remarkably reduced. Further, the divide and conquer paradigm is used, what means that the problem is divided into smaller subproblems, solution of which are easier to find and can be combined to get a solution of the original problem. In this case the problem is reduced to the problem of searching for individual elements. An important disadvantage is that it is more complicated to ensure integrity of a found match when its elements are of variable length.

RNAMot and RNAMotif fall into this category, and also we have decided to use this approach in our RNAMot2. In what follows, RNAMot and RNAMotif are described in more detail. We dedicate Section 3 to a description of our new tool RNAMot2.

2.1.1 RNAMot

RNAMot is a program devised by Gautheret et al. [1990]. The most important part of RNAMot for this thesis is Simple Scan backtracking algorithm described in the paper. We have used this backtracking algorithm in our RNAMot2, thus we will refer to it also later in Section 3. The underlying algorithm for motif structural elements matching used in RNAMot is unfortunately not described in the paper.

The Simple Scan Algorithm

The main procedure stores already matched structural elements in an array of intervals that authors call a grid. Every interval from the grid stands for the position in the sequence of a match of a strand in the motif. For example, the following motif:

$$M = H_1, S_1, H_2, S_2, H_2, S_3, H_3, s_4, H_3, S_5, H_1$$

has a corresponding grid:

$$G = i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8, i_9, i_{10}, i_{11}$$

Single-stranded elements ('s' regions) and helical elements ('H...H'), are searched in the order in which elements are searched may differ from the order in which they appear in the motif. The order is determined by a list of elements,

$$E = se_1, se_2, \ldots, se_n$$

The algorithm successively searches the sequence for each element of list E. Once an occurrence of an element is found, its position (one interval for single-stranded regions and two for a helix) is stored into its corresponding position in a grid (for a helix that are actually two positions, one for its first strand and one for the second). The algorithm continues the search for the next element in the list E. The search for any element is confined by the positions of elements already inserted in the grid. If a searched element se_i cannot be found in the sequence, than the position of the previous element se_{i-1} is removed from the grid and the algorithm continues with the search of a new occurrence of se_{i-1}. When the end of list E is reached, we have a complete match of the motif; the solution is stored and the search goes on recursively. The complete pseudocode is listed bellow, see Listing 1. The speed of the algorithm is very sensitive to the search order, thus priority should be given to searching highly specific elements such as single-stranded regions with strict sequence requirements and well-defined double-stranded (helical) regions. However, the default search order in RNAMot is constituted of helical elements of the motif ordered as they appear in the motif.

After the Simple Scan algorithm finsishes, RNAMot continues with a scoring phase. The goal of this evaluation procedure is to extract optimal solution from many options provided by the search algorithm.

Listing 1: The Simple Scan algorithm [Gautheret et al., 1990]

```
Definitions
SE           secondary structure element. Contains the following information:
             size range, sequence (optional), number of mismatches (for
             helices) and the number of nucleotides between the two strands
             (for helices). This latter item is calculated by the program.
INTERVAL two integers defining the position of a continuous sequence.
DOMAIN       list of one or two intervals (in which a given SE should be found).
GRID         array of intervals used to construct and store solutions. Each
             interval stands for a given SE.
motif        the list of all SEs.
E            the list of SEs to be searched.
seq          the sequence being analyzed.
G            a grid.
D            a domain
se           an SE.
solutions    a list of GRIDS.
match        a list of one or two intervals (describing the position of a given
             SE in the sequence). A 'helix' match is defined by two intervals
             and a 'single-strand' match by one interval.

Program SIMPLE_SCAN (seq, motif, E)
        G := [null, null, ..., null]
        solutions := 0
        find_motif (1, seq, motif, E, G, solutions)

Procedure find_motif (i, seq, motif, E, G, solutions)
        se_i := i-th SE of E
        D := get_domain (G, seq, motif, se_i)
        match := first_match (se_i, D)
        while match < > 0
                G := set_grid (G, se_i, match)
                if se_i = last SE of E
                        append G to solutions
                else
                        find_motif (i+1, seq, motif, G, E, solutions)
                match := next_match (se_i, D, match)
        end while
        G := reset_grid (G, se_i)
        end procedure

Function get_domain (G, seq, motif, se)
        Returns the domain in which se should be found (considering intervals already
        set in G and the SEs lengths in 'motif'). The returned domain is a list of
        two intervals in the case of a 'helix' SE or a list of one interval in the case
        of a 'single-strand' SE.
```

```
Function first_match (se, D)
        Returns a list of one or two intervals which describes the next occurrence of
        a se in the domain D. The returned match is a list of two intervals in the case
        of a 'helix' SE or a list of one interval in the case of a 'single-strand' SE.

Function next_match (se, D, match)
        Same as first_match except that search starts at positions defined in 'match'.

Function set_grid (G, se, match)
        Returns the grid G with interval(s) for SE se set to interval(s) in 'match'.

Function reset_grid (G, se)
        Returns the grid G with interval(s) for SE se set to null.
```

2.1.2 RNAMotif

RNAMotif is based on a two-stage algorithm devised and published by Macke et al. [2001]. In the first stage the given descriptor is converted to a search tree based on the helical nesting of its elements. Therefore this stage is called the 'compilation stage'. This tree construction is based on noting that a helix can be represented as a binary tree where the root of the tree is the helix itself, the left sub-tree is the motif between the strands of the helix and the right sub-tree is the motif that follows the helix.

A difficulty comes with pseudoknots, as they contain overlapping helices. To solve this problem the authors propose a tree composed of all the helices that form the pseudoknot as follows. The root of the structure becomes the left-most helix of the pseudoknot and it has $n-1$ interior nodes ($n \geq 4$) that represent the motifs that are contained inside all the helices that form the pseudoknot. Once again, the right-most sub-tree is the motif that follows the pseudoknot. Eventually, single-stranded elements are represented as binary trees that have an null left sub-tree (because, opposite to helices, a single-stranded element never has any interior elements) and at most one non-null right sub-tree describing the motif that follows the single-stranded region.

During the first stage also a number of consistency checks take place, for example testing that all specified lengths of the individual strands of a helix are the same or testing that if a motif element contains both explicit length specifications and implicit length constraints derived from sequence specifications, the two are compatible.

The second stage is a depth first search of the tree that was created by the compilation stage. It begins by searching the sequence for an occurrence of the left-most sub-motif of the descriptor from the left-most position. In case of several possible solutions, the algorithm examines all of them. The search algorithm is then recursively called to find all solutions of the sub-motif of the left-most interior region or, if all interior regions have been searched, the algorithm is applied to the region following the left-most motif. When a complete match is found, it is optionally evaluated by a scoring section which accepts or rejects the particular solution. Capabilities of RNAMotif scoring section are very rich and this part of RNAMotif belongs to its major strengths. Once all possible matches at a particular recursion level have been examined, the algorithm returns to the previous level and continues with the next unexamined match from the previous level. If the top level is encountered, the search is moved one position to the right. The search ends when all positions in the sequence has been examined.

The disadvantage of this approach is impossibility to determinate a search order different from the order determined by the tree structure.

Similarly to Gautheret et al. [1990] also Macke et al. [2001] gives no word on how occurrences of the motif elements are searched in the sequence.

2.2 Automaton Approach

The motif search can be also formulated as a problem in the domain of formal languages and automata. From this point of view, the motif requirements are a formal language of all its possible instances. That is, all words, that are plausible occurrences of the motif in a genomic sequence over the alphabet of nucleic acids. The search problem becomes a problem of recognizing such a language corresponding to the motif. What we need is an automaton which accepts (recognizes) the language and its computation can be efficiently implemented as a computer program.

The structure of helices (two complementary strands) implies, that the motif language is no simpler then context-free languages that can be accepted by deterministic push-down automata. But pseudoknots make a motif more complex and cause that deterministic as well as nondeterministic push-down automata are not capable of its acceptance. However, a Turing machine is surely enough, since the algorithms mentioned above can be carried out by a Turing machine.

For the purpose of developing a motif search tool more exact classification would not bring a significant benefit, because implementation of such an automaton would probably not be efficient. Nevertheless, using automaton approach to obtain a hybrid method can be worthwhile. An example of this is RNABob.

2.2.1 RNABob

RNABob is a search tool developed by Eddy [1996]. Regrettably, no paper on it has been published. We have only fragmentary information about how does it work from its short user's guide and the source code. It is said to be an efficient implementation of RNAMot using a nondeterministic finite state automaton with node rewriting rules. Further, for primary sequence matching RNABob uses a regexp subroutine and Rabin-Karp algorithm[3].

[3] a string searching algorithm using hashing

3 The New Algorithm

We named the new algorithm as RNAMot2. It uses the backtracking technique of RNAMot [Gautheret et al., 1990]. However, we have developed a new element matching algorithm, which is based on dynamic programming and is capable of finding an element match with up to specified amount of defects. If RNABob (an efficient RNAMot implementation) is used, the specified defect can only be a mismatch in primary sequence constraint or a mispairing in a helix. With RNAMot2 we bring a new feature, the ability to allow another defect. We enable to specify an insertion of a nucleotide base in the motif element. Since a deletion can be equivalently reformulated as an insertion, we this way enable deletions as well. The following text is dedicated to this new dynamic programming and RNAMot2 implementation. The backtracking needed to combine individual elements is identical to RNAMot Simple Scan algorithm described in Section 2.1.1.

3.1 Dynamic Programming Used in RNAMot2

To solve the matching problem, we have devised an individual recurrence for both single-stranded elements and helical elements. The recurrences can be solved by dynamic programming. To facilitate implementation of dynamic programming, we equivalently formulate every recurrence in a form of backward recurrence and in a form of forward implications.

3.1.1 The Recurrence for Single Strand Elements

The task is to find all occurrences of pattern P in text T with at most M mismatches and N insertions of symbol I. Insertions in a match before the first and after the last symbol of P are forbidden, moreover adjacent insertions are forbidden too. Note, that in order to keep track of positions in P and T, and of number of occurred mismatches and insertions, we create four corresponding dimensions of the recurrent function. Further, to meet the latter condition, we add the fifth dimension which stands for the fact that latest aligned symbol of T is insertion.

In what follows we define a function S:

$$S_{i,j,m,n,b} \in \{0,1\}\,;$$

$$i \in \{0..|T|\}\,, j \in \{0..|P|\}\,, m \in \{0..M\}\,, n \in \{0..N\}\,, b \in \{0,1\}$$

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

Backward recurrence

Initial condition: $\forall i \in \{0..|T|\}\ \ S_{i,0,0,0,1} := 1$

The recurrence:

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

$$S_{i,j,m,n,1} = \bigvee \begin{cases} S_{i-1,j,m,n-1,0} & \textit{iff } T[i] \text{ fits } I \text{ (insertion)} \\ S_{i,j-1,m,n,1} & \textit{iff } P[j] = \text{'}*\text{' (skip a wild card)} \end{cases}$$

Forward implications

Initial values: $\forall i \in \{0..|T|\}\ S_{i,0,0,0,1} := 1$

Implications:

$$x := (int)(T[i+1] \text{ does not fit } P[j+1])$$

$$\begin{array}{lll} S_{i,j,m,n,0} & & \Rightarrow S_{i+1,j+1,m+x,n,0} \\ S_{i,j,m,n,0} & \wedge\ P[j+1] = \text{'}*\text{'} & \Rightarrow S_{i,j+1,m,n,0} \\ \vdots & \wedge\ T[i+1] \text{ fits } I & \Rightarrow S_{i+1,j,m,n+1,1} \\ & & \\ S_{i,j,m,n,1} & & \Rightarrow S_{i+1,j+1,m+x,n,0} \\ S_{i,j,m,n,1} & \wedge\ P[j+1] = \text{'}*\text{'} & \Rightarrow S_{i,j+1,m,n,1} \end{array}$$

3.1.2 The Recurrence for Helical Elements

The task is to find all occurrences of helical pattern $P : P'$ where P, P' are strand patterns ($|P| = |P'|$ and P' is complementary to P) in text T with at most M mismatches, MP mispairings and together up to N insertions of a symbol I in both strands. Insertions are not allowed in a match before the beginning and after the end of a strand pattern. Insertions must not be adjacent nor opposite in the helix. For this purpose we extend the recurrent function for single strand elements.

We define a function H as follows:

$$H_{i,j,k,m,p,n,b} \in \{0,1\};$$

$$i,j \in \{0..|T|\}, k \in \{0..|P|\}, m \in \{0..M\}, p \in \{0..MP\}, n \in \{0..N\}, b \in \{0,1\}$$

$$H_{i,j,k,m,p,n,b} = \begin{cases} 1 & \textit{iff } P[1..k] \text{ can be aligned with a suffix } T' \text{ of } T[1..i] \text{ with} \\ & m \text{ mismatches, } P'[1..k] \text{ can be aligned with a prefix } T'' \\ & \text{of } T[j..|T|] \text{ with no mismatch, } T' \text{ and } T'' \text{ contain together} \\ & n \text{ insertions, and between } T' \text{ and } T'' \text{ are } p \text{ mispairings;} \\ & \textit{if } b = 1 \textit{ then} \text{ exactly one of } T[i],\ T[j] \text{ is insertion} \\ 0 & \textit{otherwise} \end{cases}$$

Backward recurrence

Initial condition: $\forall i, j \in \{0..|T|\}\ H_{i,j,0,0,0,0,1} := 1$

The recurrence:

$$\begin{aligned} x &:= (int)(T[i] \text{ does not fit } P[k]) \\ y &:= (int)(T[j] \text{ is not complement of } T[i]) \end{aligned}$$

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

$$H_{i,j,k,m,p,n,1} = \bigvee \begin{cases} H_{i-1,j,k,m,p,n-1,0} & \textit{iff } T[i] \text{ fits } I \text{ (insertion)} \\ H_{i,j+1,k,m,p,n-1,0} & \textit{iff } T[j] \text{ fits } I \text{ (insertion)} \\ H_{i,j,k-1,m,p,n,1} & \textit{iff } P[k] = \text{'}*\text{'}^{1} \text{ (skip a wild card)} \end{cases}$$

Forward implications

Initial values: $\forall i, j \in \{0..|T|\}\ H_{i,j,0,0,0,0,1} := 1$

Implications:

$$\begin{aligned} x &:= (int)(T[i+1] \text{ does not fit } P[k+1]) \\ y &:= (int)(T[j-1] \text{ is not complement of } T[i+1]) \end{aligned}$$

$$\begin{array}{llll} H_{i,j,k,m,p,n,0} & \wedge & T[j-1] \text{ fits } P'[k+1] & \Rightarrow H_{i+1,j-1,k+1,m+x,p+y,n,0} \\ \vdots & \wedge & P[k+1] = \text{'}*\text{'}^{1} & \Rightarrow H_{i,j,k+1,m,p,n,0} \\ \vdots & \wedge & T[i+1] \text{ fits } I & \Rightarrow H_{i+1,j,k,m,p,n+1,1} \\ \vdots & \wedge & T[j-1] \text{ fits } I & \Rightarrow H_{i,j-1,k,m,p,n+1,1} \\ & & & \\ H_{i,j,k,m,p,n,1} & \wedge & T[j-1] \text{ fits } P'[k+1] & \Rightarrow H_{i+1,j-1,k+1,m+x,p+y,n,0} \\ \vdots & \wedge & P[k+1] = \text{'}*\text{'}^{1} & \Rightarrow H_{i,j,k+1,m,p,n,1} \end{array}$$

3.2 Extension of RNABob Descriptor

The descriptor format of RNAMot2 is essentially derived from the format of RNAMot while taking over all changes brought by RNABob as described in Section 1.4. Therefore descriptors of RNABob are fully compatible with our RNAMot2. However, we add two features.

First and foremost, we add the ability to specify the number and type of allowed insertions in the structural element of a motif. The type of insertion specify which nucleotide bases can be inserted, using the characters according to IUPAC notation[4].

[1] $\forall k\ (P[k] = \text{'}*\text{'} \Leftrightarrow P'[k] = \text{'}*\text{'})$

[4] see appendix A on page 31

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

h1 0:0 G:Y
r2 0:1 NNNNN*:*NNNNN TGCA
s1 0   NN[150]
r3 0:1:1 ***NNNNNN:NNNNNN***:A TGCA
r4 0:0 NNN:NNN TGCA
s2 0   TY
r5 0:0 C:G TGCA
s3 0   HCG*Y
s4 0   N
s5 0   NNN[150]
s6 0   C*RA*

R s3 s6 s2 r4 r5 h1 r2 r3 s4 s5 s1
```

Figure 5: An example of RNAMot2 descriptor. Pay attention to definition of r3 element into which one insertion of Adenosine is allowed. The last line is reorder command specifying the search order.

Secondly, we bring back the RNAMot reorder command which specify the order in which elements are searched. If this command is absent or does not contain all elements, it is automatically supplemented by all remaining elements in order calculated by an unsophisticated heuristic which prefers longer elements with fewer wild cards.

An example of RNAMot2 descriptor can be seen in Figure 5. More descriptors are listed in Appendix B.

3.3 Implementation Notes

Implementation of the RNAMot backtracking technique is quite straightforward and is done similarly to pseudocode in Section 2.1.1. The computation of the correct search domain of an element according to already matched elements is more tricky, though. Also, to carry out the dynamic programming as stated above in an array is infeasible, thus we require a different approach.

3.3.1 Search Domain Computation

For a single strand element, *search domain* is a pair of intervals. The first interval restricts where the next match must begin while the second interval restricts where it must end. Note that these two intervals are usually not disjoint.

These intervals are computed from another two intervals which we named *upper domain bound* and *necessary coverage interval*. *Upper domain bound* is an upper estimate of the interval where a next match is allowed to occur. The *necessary coverage interval* is the interval which necessarily must be covered by a next valid match. Both *upper domain bound* and *necessary coverage interval* can be computed from the gap between the first element already matched on the left and the first element already matched on the

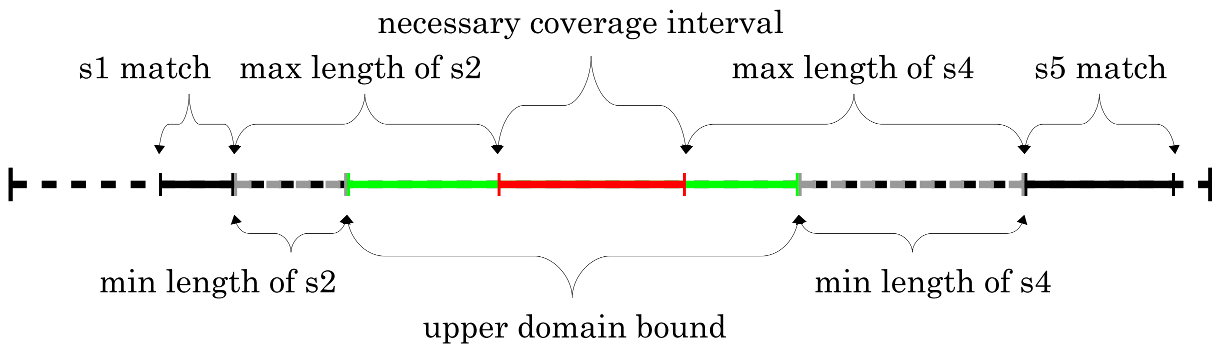


Figure 6: Ilustration of how the *search domain* is calculated. Here we have a partly matched motif composed of five single strand elements labeled from s1 to s5. Let the search order to be s1, s5, s3, s4, s2. We can see that elements s1 and s5 have already been matched and the search domain for s3 needs to be computed. Carrying out the computation stated in the text, we optain the search domain formed by the two green intervals.

right, as the gap boundaries cut by minimum and maximum length of so far unmatched elements in between, respectively (see Figure 6). When we have the *upper domain bound* and *necessary coverage interval* it is obvious that a valid match must start at a position between the beginning of the *upper domain bound* and the beginning of the *necessary coverage interval* while its end must be in the interval between the end of the *necessary coverage interval* and the end of the *upper domain bound.*

For a helical element, *search domain* is composed of domains for its two strands that are calculated as stated above. The minimum and maximum distance permitted between the strands is not taken into account here, but during the computation of the dynamic programming.

Looking for element matches with restrictions given by its *search domain* guarantees the consistency of the entire motif match, which is the ultimate purpose of search domains.

3.3.2 The Search for Structural Elements

Computing the dynamic programming in classic manner is inefficient due to the number of dimensions and sparse incidence of ones. Thus, we represent a table of dynamic programming as a list of coordinates where the ones occur. For this purpose, we use self-balancing red-black binary search tree implemented in C++ STL. Insertion of an element or finding an element in a red-black tree takes a time logarithmic in the number of the elements in the tree.

Therefore, instead of filling a table, we are rather coloring a graph. Vertices of the graph are coordinates of the table and edges are given by the corresponding function defined in Section 3.1. More accurately out-going edges are given by forward implications and incoming edges by the backward recurrence. For coloring the graph, we need to search the graph starting from the initial conditions. This is done by breadth first search in time linear in the number of vertices times the logarithmic factor coming from the red-black tree. That is $O(|T||P| \cdot \log |T||P|)$ and $O(|T||P|^2 \cdot \log |T||P|^2)$ for single-stranded element, and for helical element, respectively. If we want to reconstruct a match or find

```
Starting rnamot2: version dev1.7, May 2010
-------------------------------------------------
Database file:                database/dm_chr2R.fa
Descriptor file:              descriptors/r17.des
Search order:                 H1 H2 S2 S1
Filter out overlapping hits:  no
-------------------------------------------------


 seq-f   seq-t     name      description
------ ------ ----------- ------------
 10909   10925         chr2R
|GGTT|A|AA|ATTA|TT|AATC|
 13391   13407         chr2R
|TGGG|A|CG|ATTA|TG|TCTG|
127690  127706         chr2R
|GAGA|A|AA|ATTA|TT|TTTC|
145280  145296         chr2R
|TTAA|A|GG|ATTA|TT|TTAG|
```

Figure 7: An example of RNAMot2 output.

its beginning, we need to trace the process back to the initial condition. This operation has liner time complexity in the length of the match.

3.4 Input and Output Formats

RNAMot2 takes exactly two parameters. The first parameter is a path to a file containing motif descriptor to be searched and the second is a path to the sequence database which is a text file in FASTA format.

In FASTA format symbol '>' precedes a single-word name for the sequence, the rest of the line is taken as description. Subsequent lines contain the sequence itself, e.g.

```
>gi|173609|gb|M28984|ACARRDX A.castellani 5S ribosomal RNA
GGATACGGCCATACTGCGCAGAAAGCACCGCTTCCCATCCGAACAGCGAAGTTAAGCTGCGCGAGGCGGT
GTTAGTACTGGGGTGGGCGACCACCCGGGAATCCACCGTGCCGTATCCT
```

Output is printed on the standard output and consists of a header and of a list of matches found. The header contains elementary information about the run of RNAMot2, including identification of RNAMot2 version, used database and descriptor files, and the order in which elements were searched. Matches in the list are in the order as they were found in the database file from its beginning to its end. Every match is compounded of two lines. The first line gives the name and description (if any) of the sequence where this match occurs, the beginning position where the match starts in the sequence and the ending position where the match ends. This line is followed by a line containing the match itself, which is the substring of the sequence defined by the starting and ending positions. A symbol of pipe '|' delimits individual elements of the match. For example see Figure 7.

4 Experimental Results

In this section we evaluate RNAMot2 performance. The implementation of the algorithm is described in the previous section. We compare RNAMot2 to RNABob and RNAMotif.

Empirical tests are composed of two main categories. The first is search time of a program and the second category is the number of non-overlapping matches found (that is the maximum integer N, such that there exist N disjunct intervals containing all results). We compare the number of matches in this way because matches found by a program may overlap and the main goal of the motif search is not to find many motif matches at the same place, but rather to identify the regions where the motifs occur.

4.1 Motifs Used in the Evaluation

We decided to use both simple and more complex motifs. From the category of simple motifs we used the following motifs:

- Motif of consensus binding site for the phage R17 coat protein [Eddy, 1996, Durbin et al., 1998]. In the following text we refer to this motif as "*R17*".
- Motif of an simple artificial structure with a *bulge* [Macke et al., 2001].
- Motif of a simple *pseudoknotted structure*, it is extremely degenerated and makes a lot of matches. [Eddy, 1996].

For further evaluation of performance and features unique to RNAMot2, we used motifs used in practice by biologists. That are these motifs:

- Generalized motif of a *tRNA cloverleaf* [Eddy, 1996].
- Motif of Hepatitis delta virus ribozyme (*HDV ribozyme*) [Lupták, 2010].
- Motif of *Sassanfar aptamer* (adenosin) [Lupták, 2010]. In the following text we refer to this motif also as "*ATPapt*".

When using RNAMot2 to search for the motif of a tRNA cloverleaf and HDV ribozyme, we have also used a descriptor with reorder command to determine which elements of the motif should be searched first. Moreover, for the motif search of HDV ribozyme and ATPapt, we have also allowed for insertions allowed in some of the helices. Impact of stated changes on performance of RNAMot2 is discussed later in this section. For exact descriptors used, please see Appendix B.

4.2 Used Genomic Sequences

We have chosen real RNA and DNA sets of sequences as follows:

- *gbrna.111.0* which contains all of the sequences of the RNA section of Genbank, release 111, from April 1999. This data set contains a total number of 2264722 bases.

- *dm-chr2* which is the second chromosome of *Drosophila melanogaster*, it contains 21146708 bases. [UniGene]
- *dm-chr3* which is the third chromosome of *Drosophila melanogaster*, it contains 27905053 bases. [UniGene]

4.3 Testing Environment

All tests were run on a dual-processor computing cluster equipped with two quad-core Intel®Xeon®2.27GHz processors and 16GB of RAM. Despite the number of processing cores available, only a single core per program run was used.

On the machine was running operating system Ubuntu 9.04. Further, we have used C/C++ compiler GCC 4.3.3, GNU parser generator Bison 2.3, and fast lexical analyzer generator – FLEX 2.5.35. The last two stated programs are needed for RNAMotif to run.

We were using RNABob v2.1 released in 1996. It was compiled with the following flags:

```
CFLAGS = -O3 -D_POSIX_C_SOURCE=200112
```

The second flag was required for backward compatibility with legacy C code.

RNAMotif was used in its latest version v3.0.5 released in 2008. Compilation was performed with the same flags as for RNABob.

Our programRNAMot2 is implemented in C++ as a console application and was also compiled with optimization of highest level (`-O3`).

4.4 Methodology

Every motif was searched in every sequence database by every program. In addition, to see how the search order of motif elements and allowed insertions into a motif affect the performance of RNAMot2, we have also carried out several RNAMot2 searches with modified descriptors.

Search time of every program was measured by the built-in shell command *time*. We have used the real time spent by the process, not taking into account time spent on system calls, paging, etc.

As stated earlier in this section, the number of matches means the maximum number of non-overlapping matches in the output. We have created a simple tool to facilitate such evaluation.

4.5 The Results

All results are in joint Table 1. Subsequently, for every motif there are two graphs, one for the number of matches and one for search time.

	gbrna.111.0		dm-chr2		dm-chr3	
	# matches	time (s)	# matches	time (s)	# matches	time (s)
motif of R17						
RNABob	1	0.18	187	2.55	225	3.31
RNAMotif	1	0.07	187	0.71	225	0.98
RNAMot2	358	7.83	187	74.95	225	97.85
motif of a simple bulge						
RNABob	4875	0.69	12790	6.13	16845	8.04
RNAMotif	3401	0.22	9834	1.93	13113	2.53
RNAMot2	6185	23.93	12792	213.14	16845	291.23
motif of a simple pseudoknot						
RNABob	2390	0.92	15392	8.09	20513	10.66
RNAMotif	3557	3.31	34476	33.12	45974	43.71
RNAMot2	3116	24.99	15392	264.61	20513	335.42
motif of a tRNA cloverleaf						
RNABob	367	2.1	399	25.73	541	34.02
RNAMotif	299	2.18	0	26.46	0	34.89
RNAMot2	534	2237.07	399	22188.7	541	31980.5
RNAMot2 w/ reorder	534	8.34	399	83.58	541	112.75
motif of HDV ribozyme						
RNABob	5	51.55	7	918.95	3	1200.19
RNAMotif	32	62615.2	–	>86400	–	>86400
RNAMot2	–	>86400	–	>86400	–	>86400
RNAMot2 w/ reorder	38	1003.9	5	29421.3	3	37490.8
RNAMot2 w/ reorder & insert.	42	1014.07	11	28578.2	7	39830.4
motif of ATPapt						
RNABob	0	0.06	0	0.85	0	2.48
RNAMotif	0	0.07	0	0.64	0	0.85
RNAMot2	0	4.21	0	40.98	0	53.54
RNAMot2 w/ insert.	0	4.22	0	40.06	0	53.10

Table 1: The results

4.5.1 R17

In *gbrna.111.0* RNAMot2 has found much more occurrences of R17 than its competitors, however these additional hits were usually composed of significant amount of N characters that stands for any base. Such characters occur in the database when a part of RNA/DNA is unknown.

From the figure 9 is obvious that our RNAMot2 is quite slower, anyway its search time is acceptable.

4.5.2 Bulge

The situation with this motif is rather balanced, but RNAMotif is falling behind returning less hits.

The search time needed by RNAMot2 is considerably higher but can be assessed as real-time. From the ratio of output quality and search time, we find it better than RNAMotif.

4.5.3 Pseudoknot

RNABob and RNAMot2 are performing from the output point of view very similarly. However, RMAMotif returns lots of hits with insertions which were not specified in the descriptor. One of such extra matches is “`GTTTT CT TTGAT AGAAC GAAA GTCAG`” at posititon 824 in *Amblyomma americanum 18S rRNA* sequence from *gbrna.111.0* database, whose alignment to the seqence is as follows:

```
GTTTT CT TTGAT CAAGAAC GAAA GTCAG
GTTTT CT TTGAT --AGAAC GAAA GTCAG
```

RNAMotif has allowed insertion of “`CA`” into the second helix of the pseudoknot, what is not allowed by the descriptor as can be seen in Figure 24 (Appenix B).

Despite the fact that the insertions are not allowed by the descriptor, this behavior is probably desired by RNAMotif authors, as they want to find as many as possible reasonable matches (with little defects according to descriptor). RNAMotif then rely on its robust scoring section.

The graph of search time in Figure 13 proves RNAMot2 to be again slower, however with more demanding motif the gap ratio is showing to decrease.

4.5.4 tRNA

Again, RNAMot2 and RNABob returns almost the same results. We have no explanation to what happened with RNAMotif that it did not found a single match in chromosomes of *Drosophila melanogaster*.

When it comes to more real and complex motif, the situation is about to change. For the first time in our set of tests we have used reorder command of RNAMot2 to help the program to decide which elements of the motif should be search first. As we can see in Figure 15, the search time of RNAMot2 has been significantly reduced and is not very falling behind the time of RNABob and RNAMotif.

4.5.5 HDV ribozyme

This motif is the most demanding motif from our test. RNAMot2, without good search order specified by user in descriptor, was not able to finish in 24 hours. Also, RNAMotif runs out of the time limit when searching in chromosomes of *Drosophila melanogaster* that are roughly 10 times longer than the length of gbrna.111.0 database. Therefore zero number of hits in Figure 16 means that a program did not managed to finish in 24 hour time limit.

Because this motif may occur with an insertion of Adenosine into one of its helices, we run RNAMot2 also with modified descriptor allowing one insertion of this type.

When looking at the number of discontinued number of hits found, RNAMot2 is doing well and with allowed insertion it finds considerably more plausible matches.

From search time point of view RNABob is performing the best by far. RNAMot2 with reorder command finished within the time limit with time from ~8 to ~11 hours on chromosomes of *Drosophila melanogaster*. So, good search order shows to be crucial for RNAMot2 time performance, while an insertion to the motif had not significant influence on the search time.

4.5.6 Sassanfar aptamer

For this motif, we have graph only for search time because none of the programs found a match in the databases. From this test-run, we can see that if elements of a motif are not occurring often in a database then every program is quite fast.

4.6 Results Summary

Our tests have shown that RNAMot2 and RNABob usually find the same set of matches when searching the same descriptor. But with the new feature enabling to allow insertions in the motif elements is RNAMot2 even more sensitive. Due to the feature more of plausible motif occurrences, such with specified mutations, can be found. This fact documents the test with motif of HDV ribozyme. RNAMot2 is also more sensitive then RNAMotif. This is important when searching for new genes.

The drawback of our RNAMot2 is speed. In this field RNAMot2 sometimes considerably falls behind the others. However, also our tests reveal that the time performance can be improved significantly. To this we dedicate the next section.

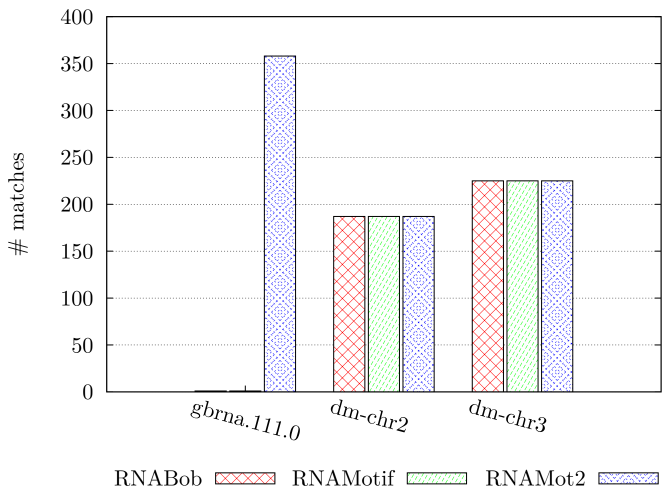


Figure 8: r17 – matches

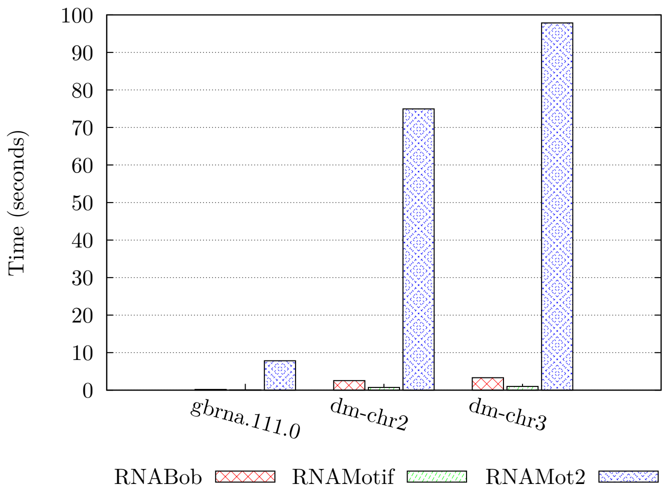


Figure 9: r17 – time

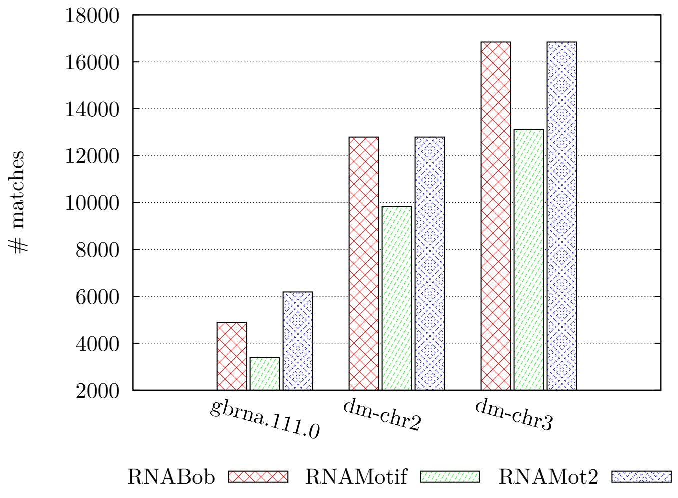


Figure 10: bulge – matches

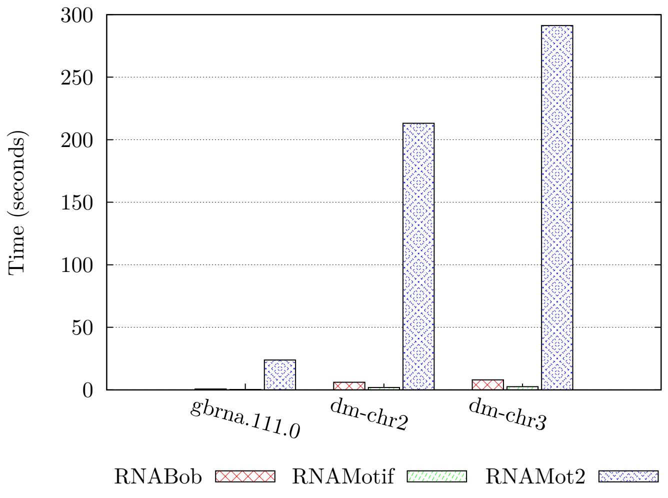


Figure 11: bulge – time

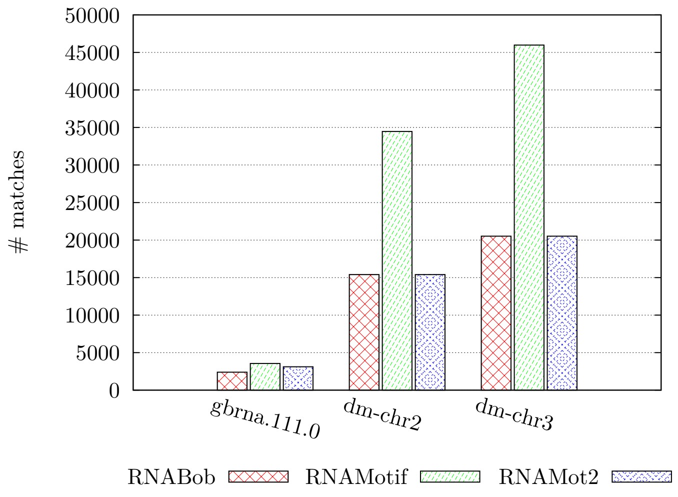


Figure 12: pseudoknot – matches

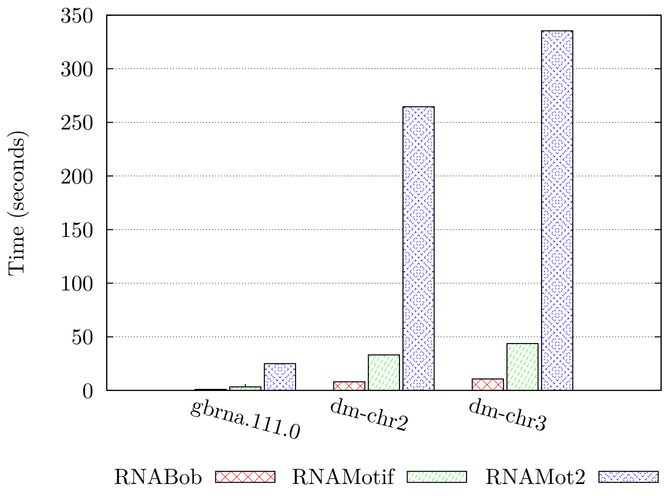


Figure 13: pseudoknot – time

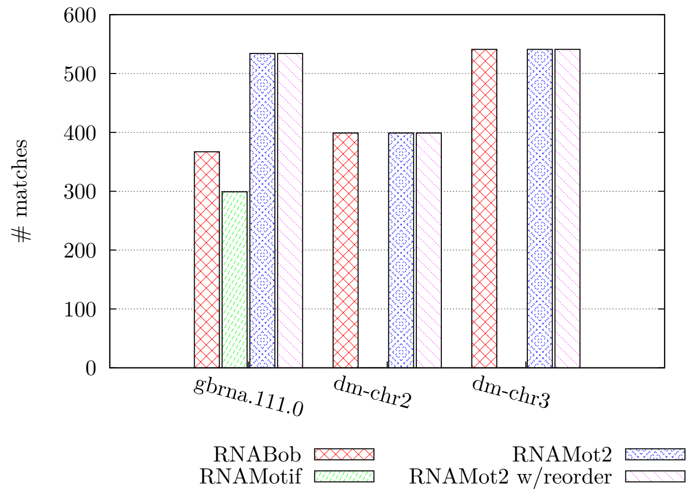


Figure 14: tRNA – matches

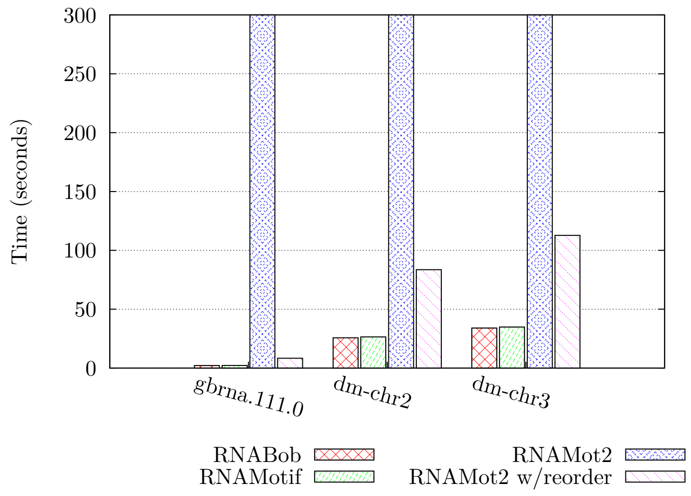


Figure 15: tRNA – time

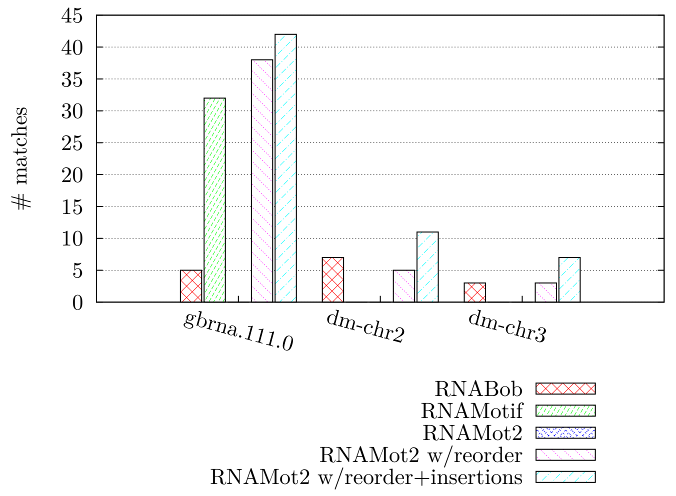


Figure 16: HDV – matches

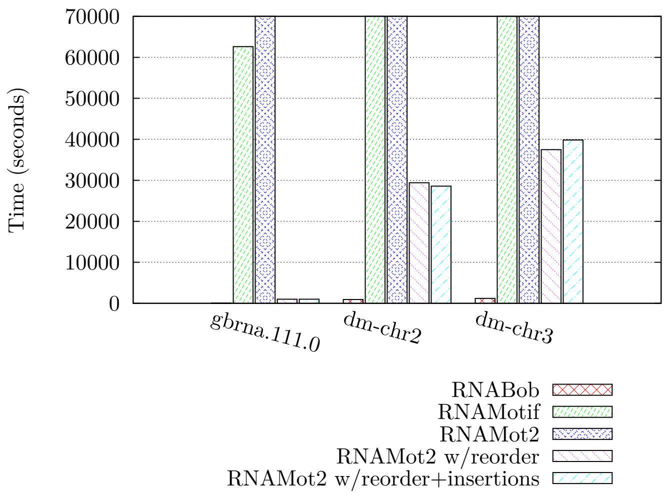


Figure 17: HDV – time

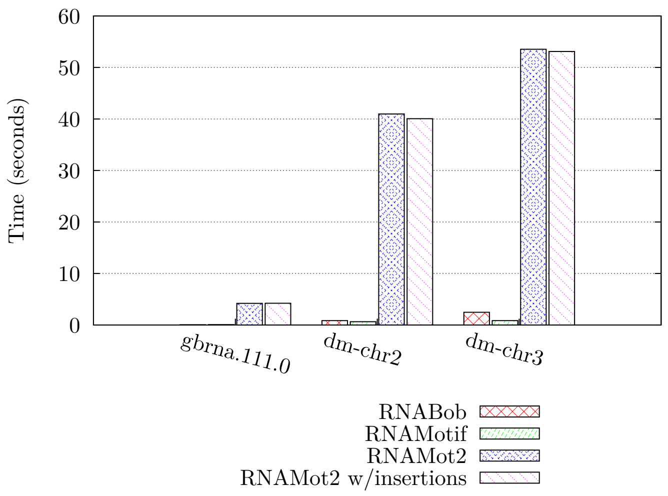


Figure 18: ATP – time

5 Future Work

The experimental results stated in the previous section imply that our RNAMot2 needs some improvement in the area of time performance. Here, we present several ideas which should help to reach the desired progress in this field.

5.1 Suitable Search Order

In the previous section, we have shown that the search order of individual elements can significantly affect running time. By a good search order we want to achieve that as little as possible partly matches which cannot be spread to full matches advance to next level of the backtrack search. In other words, we want to cut false branches of the backtracking as soon as possible and spend as little time as possible at places where it is not to match the entire motif.

Thus we should first search for highly specific elements whose random occurrence in a sequence is unlikely. For example, single strand elements with strict primary sequence requirements and well-defined helical elements are often highly specific. On the other hand, elements of very variable length and/or motif, should be searched the last, since they can actually help to fill the gap between adjacent elements.

Elements of high specificity and a good search order is a matter of heuristic involving statistics. A research of this issue can bring significant speed-up of the search for complex motifs with many elements. Current implementation enables a user to specify the search order, but in the future choice of a good ordering should be automated.

5.2 Filtering

Even with a good search order of elements, the very first elements are searched in the entire sequence (which is usually very large) or in its large portion. The time complexity of the dynamic programming for single-stranded elements is basically linear in the length of the sequence and pattern. For helical elements the dependence on the length of a pattern is quadratic. Also, we have to take into account the time complexity of operations over the underlying data structure which are logarithmic in case of the used balanced search tree.

We have to pay this price if we want to execute an absolutely correct element searching, but may be in the first step of the search, we would like to prefer speed in favor of the quality. Here comes the idea of filtering. The goal of filtering is to sort out as many segments of the sequence in which the motif occurrence is not possible, while the time used is shorter than the time of the correct search. Only the surviving segments of the sequence are searched by the correct more-time-consuming search.

Such filtration can be done in many different ways. Here, we present ideas based on algorithms for approximate string matching [Gusfield, 1997].

The first method is devised by Baeza-Yates and Perleberg [1992] and have been used in their BYP method for k-difference approximate string matching. It is based on the following lemma:

> Let $r = \left\lfloor \frac{|P|}{k+1} \right\rfloor$, and partition P into consecutive r-length regions. Suppose P matches a substring T' of T with at most k differences. Then T' must contain

at least one interval of length r that exactly matches one of the r-length regions of the partition of P.

On this lemma we can base a filtering method also for our problem. First, we choose from the searched motif M elements with well defined primary sequence constraints. Denote this constraints as patterns P_i. For such P_i we know the number k_i of allowed defects[5]. Secondly, by partitioning the P_i according to the lemma we are getting a set of partitions S_i. Let $S = \bigcup_i S_i$. Than the lemma implies that an occurrence of the motif M must contain all the elements of S as non-overlapping substrings.

This is a standard searching problem that can be solved by well-known liner-time algorithms, such as Aho-Corasik or Boyer-Moore extended to set matching. After applying the filter only those parts of the sequence which contains occurrences of all strings from S close together and in correct order would be searched by our algorithm.

In addition to filtering methods used in algorithms for approximate string matching, also the algorithms itself can be used for filtering as well. Since there are linear and subliner expected time algorithms [Gusfield, 1997], the idea above can be adopted. However, the algorithms usually cannot be easily modified to cope with IUPAC nucleic acids notation[6] used to describe primary sequence constraints. The problem is caused by characters with ambiguous meaning. The problem arise, for example, when a suffix tree is used.

5.3 Faster Data Structure

In present RNAMot2 implementation, as stated in Section 3.3.2, usage of Red-black trees adds a logarithmic factor to the time complexity of the dynamic programming. In expected case this factor can be eliminated by changeover to hash maps whose operations are of constant time complexity in expected case.

5.4 Trade Memory for Speed

There is also possibility of trading more memory for speed. One way of doing this is to do more caching. Another method can be the Four-Russians speed-up [Arlazarov et al., 1970, Gusfield, 1997]. This approach leads to speed-up of the dynamic programming. The general idea lies in precalculation of a function for blocks of original dynamic programming table. The table is then filled in block after block rather than cell by cell.

5.5 Parallelism

The previous statements apply to algorithm itself, nevertheless that is not the only point to which optimization applies. In practice, the real time spent on a run is what counts when speaking about time performance. Modern machines bring another possibility to cut the time. Since a searched sequence, if significantly longer than a search motif, can be easily divided into shorter sequences (with appropriate overlap) then the problem can be divided into smaller instances which can run in parallel. Taking advantage of today's multi-core systems may lead to a significant speed-up.

[5]that is the number of allowed insertions and mismatches in the corresponding element

[6]see Appendix A on page 31

Conclusion

Applications of computer science is usually have an interdisciplinary aspect. In this work we consider a problem, where informatics and biology meet in bioinformatics. We have been dealing with a problem proposed by biological science which can be exactly formulated in the language of computer science as a pattern matching problem. Chemical principles in nature cause the searched patterns – RNA motifs – to be decomposable into elements that contain internal dependencies. Moreover, biological variability, as for example mutations and similar chemical properties of some different nucleotide bases or base pairs, make the definition of an RNA motif more abstract and also leads to requirements where complex error patterns are allowed. The ability to meaningfully define admissible errors is important, as only matches with variations that do not change biochemical functionality of the motif substantially are desired. All the conditions make the search problem more difficult, however the search must be efficient enough to enable motif search in extensive genomic databases.

In this work, we have designed a new tool facilitating the search. We named the tool RNAMot2. RNAMot2 search algorithm basically consists of two parts. One part is an element matching algorithm based on dynamic programming and the second part is a backtracking algorithm to combine the individual motif element matches in a match of the entire motif. We adopted a backtracking algorithm from existing tool RNAMot [Gautheret et al., 1990].

As stated above, the possibility to search with errors is very important for finding plausible matches. For this purpose, besides errors definable by current tools, we have introduced possibility to allow insertions of a nucleotide base (of defined type) into the motif element sequence. Furthermore, since a deletion of a nucleotide base (of defined type) in the motif element sequence can be equivalently reformulated as an insertion, we brought by this way support for deletions, too. In order to enable such new declaration, we have supplemented the RNABob motif descriptor and we have proposed a new dynamic programming algorithm for element search.

We tested RNAMot2 and compared its performance with RNABob and RNAMotif. Our empirical results have proven that RNAMot2 can be considerably more sensitive due to allowing insertions and deletions. Our tool has turned out to be significantly slower, however, tests on descriptors with good reorder command have revealed one of possible ways to reduce RNAMot2 search time. We also propose more ways to accomplish further time reduction, for example sequence prefiltering and code optimization.

A Appendix: The IUPAC Notation

	Symbol	Meaning	Mnemonic
DNA Bases	A	Adenosine	Adenosine
	C	Cytosine	Cytosine
	G	Guanine	Guanine
	T	Thymine	Thymine
Ambiguity Characters	R	G + A	puRine
	Y	T + C	pYrimidine
	S	G + C	Strong interactions (3H bonds)
	W	T + A	Weak interactions (2H bonds)
	K	G + T	Keto
	M	A + C	aMino
	D	G + T + A	Not-C (D follows C in alphabet)
	H	T + A + C	Not-G (H follows G)
	B	G + T + C	Not-B (B follows A)
	V	G + A + C	Not-T or U (V follows U)
	N	G + A + T + C	aNy

Table 2: The IPUAC nucleic acid notation

B Appendix: Used Descriptors

In this appendix we present exact listings of all descriptors of motifs which were used for performance assessment of our RNAMot2 against RNABob and RNAMotif.

B.1 R17

This motif is consensus binding site for the phage R17 coat protein. It is composed of a stem with a single-base A bulge and a 4-base loop. [Eddy, 1996]

```
h1 s1 h2 s2 h2' h1'

h1 0:0 NNNN:NNNN
h2 0:0 NN:NN
s1 0   A
s2 0   AUYA
```

Figure 19: Listing of R17 descriptor for RNABob and RNAMot2

```
parms
        wc += gu;

descr
        h5( tag="1", len=4)
                ss( len=1, seq="A" )
                h5( tag="2", len=2)
                        ss( len=4, seq="^AUYA$" )
                h3( tag="2" )
        h3( tag="1" )
```

Figure 20: Listing of R17 descriptor for RNAMotif

B.2 Bulge

This is a motif of an simple artificial structure with a bulge. [Macke et al., 2001]

```
h1 s1 h2 s2 h2' s3 h1'

h1 0:0 NNN:NNN
s1 0 N**
h2 0:0 NNN:NNN
s2 0 GNRA
s3 0 N**
```

Figure 21: Listing of descriptor of a bulge for RNABob and RNAMot2

```
parms
        wc += gu;

descr
        h5( len=3, strict=3 )
                ss( minlen=1, maxlen=3 )
                h5( len=3 )
                        ss( seq="^GNRA$" )
                h3
                ss( minlen=1, maxlen=3 )
        h3
```

Figure 22: Listing of descriptor of a bulge for RNAMotif

B.3 Pseudoknot

This is a motif of a simple pseudoknotted structure. It is not modeled on anything in particular, and is extremely degenerated and makes a lot of matches. [Eddy, 1996]

```
h1 s1 h2 h1' s3 h2'

h1  0:0  NNNNN:NNNNN
h2  0:0  NNNNN:NNNNN

s1  0    N**
s3  0    NNN**
```

Figure 23: Listing of descriptor for a simple pseudoknotted structure for RNABob and RNAMot2

```
parms
        wc += gu;

descr
        h5( tag="1", len=5 )
                ss( minlen=1, maxlen=3 )
        h5( tag="2", len=5 )
        h3( tag="1" )
                ss( minlen=3, maxlen=5 )
        h3( tag="2" )
```

Figure 24: Listing of descriptor for a simple pseudoknotted structure for RNAMotif

B.4 tRNA

Generalized descriptor of a tRNA cloverleaf. However, it does not find them all. [Eddy, 1996]

```
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
```

Figure 25: Listing of tRNA descriptor for RNABob and RNAMot2

```
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

R s8 h4 s6
```

Figure 26: Listing of tRNA descriptor for RNAMot2 using reorder command

```
parms
        wc += gu;

descr
        h5(minlen=7,maxlen=7)
                ss( seq="^TN$" )
                h5(minlen=3,maxlen=4)
                        ss(minlen=4,maxlen=14)
                h3
                ss(len=1)
                h5(minlen=5,maxlen=5)
                        ss(minlen=6,maxlen=7)
                h3
                ss(minlen=2,maxlen=22)
                h5(len=5)
                        ss(len=7, seq="^TTC")
                h3
        h3
        ss(seq="^NCCA$")
```

Figure 27: Listing of tRNA descriptor for RNAMotif

B.5 HDV ribozyme

This is a loose form of Hepatitis delta virus (HDV) ribozyme motif. [Lupták, 2010]

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

h1 0:0 G:Y
r2 0:1 NNNNN*:*NNNNN TGCA
s1 0   NN[150]
r3 0:1 ***NNNNNN:NNNNNN*** TGCA
r4 0:0 NNN:NNN TGCA
s2 0   TY
r5 0:0 C:G TGCA
s3 0   HCG*Y
s4 0   N
s5 0   NNN[150]
s6 0   C*RA*
```

Figure 28: Listing of HDV ribozym descriptor for RNABob and RNAMot2

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

h1 0:0 G:Y
r2 0:1 NNNNN*:*NNNNN TGCA
s1 0   NN[150]
r3 0:1 ***NNNNNN:NNNNNN*** TGCA
r4 0:0 NNN:NNN TGCA
s2 0   TY
r5 0:0 C:G TGCA
s3 0   HCG*Y
s4 0   N
s5 0   NNN[150]
s6 0   C*RA*

R S3 S6 S2 r4 r5 h1 r2 r3 s4 s5 s1
```

Figure 29: Listing of HDV ribozym descriptor for RNAMot2 using reorder command

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

h1 0:0 G:Y
r2 0:1 NNNNN*:*NNNNN TGCA
s1 0    NN[150]
r3 0:1:1  ***NNNNNN:NNNNNN***:A TGCA
r4 0:0 NNN:NNN TGCA
s2 0    TY
r5 0:0 C:G TGCA
s3 0    HCG*Y
s4 0    N
s5 0    NNN[150]
s6 0    C*RA*

R s3 s6 s2 r4 r5 h1 r2 r3 s4 s5 s1
```

Figure 30: Listing of HDV ribozym descriptor for RNAMot2 using reorder command and insertions

```
descr
      h5(tag="h1r2", minlen=6, maxlen=7, seq="^G")
            ss( minlen=2, maxlen=152 )

            h5(tag="r3", minlen=6, maxlen=9)
            h5(tag="r4", minlen=3, maxlen=3)

                  ss(seq="^TY$")

                  h5(tag="r5", minlen=1,maxlen=1,seq="C")
                        ss(seq="^HCG$")
                        ss(minlen=1,maxlen=2,seq="Y$")
                  h3(tag="r4")
      h3(tag="h1r2")
      h3(tag="r5")

      ss( minlen=4, maxlen=154 )
      ss( minlen=1, maxlen=2, seq="^C")
      ss( minlen=2, maxlen=3, seq="^RA")

      h3(tag="r3")
```

Figure 31: Listing of HDV ribozym descriptor for RNAMotif

B.6 Sassanfar aptamer (adenosin)

More strict form of Sassanfar aptamer (adenosin) motif. [Lupták, 2010]

```
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
```

Figure 32: Listing of Sassanfar aptamer (adenosin) descriptor for RNABob and RNAMot2

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

h1 0:0:1 *****:*****:A
r2 0:0 NNNN:NNNN TGCA
r3 0:0 CNNN:NNNG TGCA
h4 0:0:1 *****:*****:A

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

Figure 33: Listing of Sassanfar aptamer (adenosin) descriptor for RNAMot2 using reorder command and insertions

```
descr
        h5(minlen=4,maxlen=9)
                ss( seq="^GGAAGAAACTG$" )
                h5(minlen=4,maxlen=9, seq="CNNN$")
                        ss(minlen=3,maxlen=20)
                h3
                ss( seq="^G$" )
        h3
```

Figure 34: Listing of Sassanfar aptamer (adenosin) descriptor for RNAMotif

References

V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradzev. On economic construction of the transitive closure of a directed graph. *Dokl. Acad. Nauk SSSR*, 194: 487–488, 1970.

R. Baeza-Yates and C. Perleberg. Fast and practical approximate string matching. In *Combinatorial Pattern Matching*, pages 185–192. Springer, 1992.

P. Bengert and T. Dandekar. Riboswitch finder–a tool for identification of riboswitch RNAs. *Nucleic acids research*, 32(Web Server Issue):W154, 2004.

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

A. Brazma, H. Parkinson, T. Schlitt, and M. Shojatalab. A quick introduction to elements of biology - cells, molecules, genes, functional genomics, microarrays. "http://www.ebi.ac.uk/microarray/biology_intro.html", European Bioinformatics Institute, 2001.

Tzu-Hao Chang, Hsien-Da Huang, Tzu-Neng Chuang, Dray-Ming Shien, and Jorng-Tzong Horng. RNAMST: efficient and flexible approach for identifying RNA structural homologs. *Nucl. Acids Res.*, 34:W423–428, 2006.

R. Durbin, S.R. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998. ISBN 0521629713.

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

D. Gautheret, F. Major, and R. Cedergren. Pattern searching/alignment with RNA primary and secondary structures: an effective descriptor for tRNA. *Comput Appl Biosci*, 6(4):325–31, 1990. ISSN 0266-7061.

A. D. George and S. A. Tenenbaum. Informatic resources for identifying and annotating structural RNA motifs. *Mol Biotechnol*, 41(2):180–93, 2009. ISSN 1073-6085.

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

D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, 1997.

A. Lupták. personal communication, 2010.

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

P. Schattner, A.N. Brooks, and T.M. Lowe. The tRNAscan-SE, snoscan and snoGPS web servers for the detection of tRNAs and snoRNAs. *Nucleic acids research*, 33(Web Server Issue):W686, 2005.

UniGene. National Center for Biotechnology Information database. "`ftp://ftp.ncbi.nih.gov/repository/UniGene/Drosophila_melanogaster/`".

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