Media Engineering and Technology Faculty
German University in Cairo

Inferring RNA Stem-Loop Descriptors from Multiple Sequence-structure Alignments for an Indexed-based RNA Search Method

Bachelor Thesis

Author:	Baher S.A. Salama
Supervisor:	Prof. Dr. Rolf Backofen
	Dr. Sebastian Will
	Steffen Heyne
	Michael Beckstette
Reviewers:	Prof. Dr. Slim Abdennadher
Submission Date:	5 September, 2009

This is to certify that:

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

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

Baher S.A. Salama
5 September, 2009

Acknowledgement

I would like to thank Prof. Dr. Rolf Backofen for inviting me to do this bachelor thesis in his Department of Bioinformatics at Albert-Ludwigs Univerisity in Freiburg. I would also like to thank my supervisor Steffen Heyne for his continuous, daily support and guidance. In addition, I would like to thank Dr. Sebastian Will and Michael Beckstette for their support and their interest in my thesis work. I would also like to thank Prof. Dr. Slim Abdennadher for arranging me this opportunity.

Abstract

Since the discovery of the variety of functional roles performed by non-protein-coding RNA (ncRNA), the search for homologous RNAs has been a problem of great interest in the field of bioinformatics. This thesis presents a new technique that uses an index-bases search tool to perform RNA homology search. A multiple sequence alignment of an RNA family is used to generate search patterns/descriptors for an affix-array-based search tool which uses the descriptors to search a nucleotide sequence database for new members of the given RNA family. The results of the search are evaluated and compared to results produced by other famous tools that perform a similar task using different techniques. In addition, the thesis introduces two extensions that were developed for the index-bases search tool used.

Contents

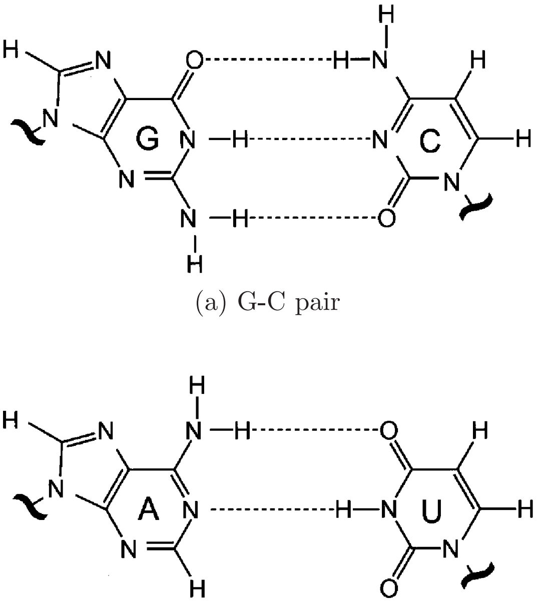
Chapter 1

Introduction

The discovery of multiple functional roles for non protein-coding ribonucleic acids (RNA) has made them the focus of much research in the recent years. Non-coding RNAs (ncRNA) have been found to be responsible for many key functional roles in living cells. Some examples of such roles are transcriptional regulation, RNA processing and modification, chromosome replication, mRNA stability and replication and protein degradation [21]. The problem of detecting homologous sequences as well as conserved motifs in ncRNAs has been of great interest in the field of bioinformatics. Several methods have been proposed for detecting homologues, finding conserved regions within families, predicting secondary structures, performing multiple sequence and structure alignments and performing sequence and structure searches for ncRNAs in nucleotide sequence databases. Many databases have been built solely for storing, grouping, and annotating ncRNA families.

1.1 Ribonucleic acids

A ribonucleic acid (RNA) molecule is a biopolymer composed of a chain of nucleotides. A single nucleotide is composed of a phosphate group, a ribose sugar and a nitrogenous base. There are four types of bases found in RNA which are Adenine (A), Guanine (G), Cytosine (C) and Uracil (U). Deoxyribonucelic Acids (DNA) are also composed of linked nucleotide chains. However, DNA nucleotides contain deoxyribose sugar instead of ribose and Thymine (T) base instead of Uracil. This project focuses mainly on RNAs. Throughout the thesis, the bases will normally be referred to using their single-letter abbreviations (shown in parentheses).



(a) G-C pair

(b) A-U pair

Figure 1.1: Watson-Crick base pairs. Figures taken from [24]

Base pairing and secondary structure

The formation of hydrogen bonds between the bases of different nucleotides in RNA molecules is a phenomenon referred to as *base pairing*. The most common base pairs occur between Adenine and Uracil; and between Guanine and Cytosine, these are known as *Watson-Crick pairs* (Figure 1.1). Other types of base pairs, such as Guanine-Uracil, can also occur but are less frequent.

The formation of base pairs between two different nucleotides in the same RNA molecule causes the molecule to fold forming what is referred to as *secondary structure*. A correct secondary structure is crucial for an RNA to perform its function correctly. The most common structural element is the *stem-loop* or *hairpin*.

Coding vs. non-coding RNA

The role of protein coding RNA is to transfer the genetic data encoded in DNA to be synthesised into proteins. This process takes place in 2 phases: transcription and translation. In the transcription phase, a sequence of DNA is encoded on a corresponding sequence of RNA. In the translation phase, the formed RNA sequence is used to construct the protein.

Non-coding RNAs, on the other hand, are not translated into proteins. Instead, they perform a variety of catalytic, regulatory and other functional roles. Like protein-coding RNA, non-coding RNA is transcribed from a region of DNA in a transcription process. This project is concerned mainly with non-coding RNA.

Representation of primary sequence

Due to the asymmetric linkage of RNA nucleotides, the “direction” of the nucleotide sequence needs to be defined. The end of an RNA molecule linked to the 5' carbon atom of the ribose sugar is called the *5' end* while the end linked to the 3' atom is called the *3' end*. Conventionally, the 5' end is regarded as the “starting point” of the RNA strand. Thus the RNA molecule can be represented as a string over the alphabet $\Sigma = \{A, G, C, U\}$, where the letters represent the abbreviation of the bases stated above. This sequence is called primary sequence.

Definition 1 (Primary sequence)**.**
Let Σ be the nucleotide alphabet $\{A, G, C, U\}$. The primary sequence S is defined as $S = \langle s_1, s_2, s_3, \ldots, s_n \rangle$, where $n \in \mathbb{N}$ and $s_i \in \Sigma$, for $1 \leq i \leq n$.

Througout the thesis, $|S|$ will denote the length of the sequence S, S_i will denotes the i^{th} element of the sequence S, and S_{ij} will denote the substring of S between i and j inclusive, where $1 \leq i \leq j \leq |S|$. The following definition defines when two sequences are considered *equal*.

Definition 2 (Sequence equality)**.** *Two sequences X and Y are equal if-and-only-if $|X| = |Y| = n$ and $X_i = Y_i$ for all $1 \leq i \leq n$.*

Representation of secondary structure

To represent base pairings between the nucleotides of a sequence, the secondary structure of an RNA sequence is defined as follows:

Definition 3 (Secondary structure)**.**
Given a primary sequence S, the secondary structure T over S is defined as follows:
$T = \{(i, i') : S_i$ is paired with $S_{i'}\}$.

For the scope of this work, valid secondary structures must satisfy the following:

1. $\forall_{(i,i') \in T} : 1 \leq i < i' \leq |S|$.

2. $\forall_{(i,i'),(j,j')\in T} : (i \neq i')$ *and* $(j \neq j')$ *and* $(i = j \iff i' = j')$.

3. $\neg\exists_{(i,i'),(j,j')\in T} : i < j < i' < j'$. *i.e. pseudo-knots are not allowed.*

In practice, the secondary structure is represented in dot-bracket notation which is defined as follows:

Definition 4 (Dot-bracket notation)**.** *Given a sequence S with secondary structure T,* dot-bracket notation *represents the secondary structure as a sequence B with the same length as S over the alphabet* $\{$`(`, `)`, `.`$\}$, *such that* $B = \langle b_1, b_2, \ldots, b_n\rangle$, *where* $n = |S|$ *and* $b_i =$ '`(`' *and* $b_{i'} =$ '`)`' *if* $(i, i') \in T$, *otherwise* $b_i =$ '`.`', *for* $1 \leq i < i' \leq n$

Hairpin structure

Definition 5 (Hairpin)**.** *Given a primary sequence S and a secondary structure T. The subsequence S_{ab} is a hairpin structure if:*

1. $(a, b) \in T$, *and*

2. $\neg\exists_{(i,i'),(j,j')\in T} : (i > a) \wedge (j' < b) \wedge (j > i')$, *i.e. no branchings.*

Definition 6 (Maximal hairpin)**.** *Given a primary sequence S and a secondary structure T such that S_{ab} is a hairpin. S_{ab} is a* Maximal hairpin *if there is no proper supersequence of S_{ab} in S that is a hairpin, i.e. there does not exist $a' \leq a$ and $b' \geq b$ such that $(b' - a') > (b - a)$ and $S_{a'b'}$ is a hairpin according to definition 5.*

Throughout this thesis, the term *hairpin* will used to refer to the maximal hairpin, unless otherwise noted.

Many of the definitions above are inspired by [12].

1.2 RNA families

Non-coding RNAs are categorised into families according to functional or structural similarities. RNA sequences within the same family usually share certain regions or motifs. Finding these conserved regions or motifs in RNA families as well as finding RNA family member within genomes are problems of great interest in bioinformatics. Several models, techniques and tools have been developed for solving these problems. Some of these tools are introduced in the next section. The following are some common examples of RNA families:

Transfer RNA (tRNA): RNA molecules that contribute to the RNA translation process by binding to specific amino acids and transferring them to ribosomes to be assembles into polypeptide chains. The secondary structure of tRNA is characterized by three hairpin loops.

RNase P: a ribozyme (RNA enzyme) that performs post-transcriptional modification of tRNA [7].

Small nucleolar RNA (snoRNA): RNA molecules that guide post-transcriptional chemical modifications of tRNAs.

MicroRNA (miRNA): a family of short (~22 nucleotides long) ncRNAs. miRNAs perform regulation of gene expression [4].

1.3 RNA alignments

Definition 7 (Alignment)**.**
Given a set A' of sequences over an alphabet Σ, an alignment of A' is an $m \times n$ matrix A over alphabet $\Sigma_A = (\Sigma \cup \{\_\})$, where $m = |A|$. The rows of the matrix represent sequences each of length n, where the i^{th} sequence is represented as $S_i = \langle A_{i,1}, A_{i,2}, \ldots, A_{i,n} \rangle$, where $A_{i,j} \in \Sigma_A$, for $1 \leq j \leq n$, $1 \leq i \leq m$. The following conditions also hold:

1. *for every sequence s' in A' there is a corresponding sequence s in A such that removing all '_' characters from s produces s'.*

2. $\neg\exists_{1 \leq j \leq n} : \forall_{1 \leq i \leq m} : A_{i,j} =$ '_', *i.e. no column that contains only gaps.*

For an alignment A with dimensions $m \times n$, the common length n will sometimes be denoted as the *aligment length* or *length*$[A]$. The number of sequences in the alignment m will be denoted as $|A|$. If $|A| = 2$ the alignment is called a *pairwise alignment*, while if $|A| > 2$ the alignment is called a *multiple alignment*. The '_' symbol in the definition above denotes a gap in the alignment, which is represented in text format as `.` or `-`. Figure 1.3 on page 11 shows an example of a multiple alignment with secondary structure annotation.

Definition 8 (Alignment column)**.** *Given an alignment A over a finite alphabet Σ_A. The j^{th} column of the alignment is defined as a sequence: $C_{A,j} = \langle A_{1,j}, A_{2,j}, \ldots, A_{m,j} \rangle$, where $m = |A|$ and $A_{i,j} \in \Sigma_A$, for $1 \leq i \leq m$.*

Definition 9 (Alignment column sequence)**.**
Given an alignment A. The column sequence of the alignment C_A is defined as $C_A = \langle C_{A,1}, C_{A,2}, C_{A,3}, \ldots, C_{A,n} \rangle$, where $n = length[A]$ and $C_{A,i}$ is the i^{th} column of alignment A, for $1 \leq i \leq n$.

Definition 10 (Subalignment)**.**
Given two alignments A and B over the same alphabet, B is a subalignment *of A only if C_B is a subsequence of C_A, where C_X is the column sequence of alignment X.*

1.4 RNA homology search and related work

1.4.1 Alignment tools

This section introduces some sequence and/or structure alignment tools that are used for searching and grouping homologous RNA.

LocARNA [26]

LocARNA introduces an efficient variant of Sankoff's algorithm for simultaneous alignment and secondary structure prediction (see [18]). For sequences of length n, LocARNA reduces the time and space complexities of pairwise alignment from $O(n^6)$ and $O(n^4)$ in Sankoff's algorithm to $O(n^4)$ and $O(n^2)$, respectively. This significant enhancement makes it efficient to use LocARNA for computing all pairwise alignments of a large set of RNA sequences. The resulting alignment scores can then be used to cluster RNAs according to sequence and structure similarities.

Infernal [16]

Infernal takes as input a multiple sequence alignment of a group of related RNAs with secondary structure annotation. It uses the alignment to construct a covariance model (CM) (see [6]) that describes probabilistically the consensus sequence and structure of the given alignment. It then uses the constructed CM to search nucleotide sequence databases for homologous RNA molecules.

1.4.2 Motif discovery and search tools

RNAProfile [17]

RNAProfile takes as input an unaligned set of RNA sequences expected to share a common motif along with the expected number of hairpin loops in the shared motif. RNAProfile highlights the shared motif regions in the given set of RNA molecules. This information can be used by other tools to search for RNAs that exhibit similar structural or functional properties to the given RNA molecules.

ERPIN [9]

The ERPIN software takes an RNA sequence alignment with secondary structure information and constructs two probabilistic profiles one for sequence information and one for base pairing information. It then uses the constructed profiles to perform a database search for sequences homologous to those given in the input. The search uses dynamic programming alignment of the generated profiles with the searched sequence.

RNAMotif [14]

RNAMotif takes as input a complex description of RNA secondary structure. The structure description language allows the user to specify allowed base pairings, structural elements, scoring schemes and other detailed specifications. RNAmotif then uses the description to search a nucleotide sequence database for sequences that satisfy the given structural constraints.

RNAalifold [5]

RNAalifold takes as input a multiple RNA sequence alignment and computes its consensus secondary structure.

1.5 RNA family databases

There exist several online databases specialised for storing, grouping and annotating ncRNAs. Such databases provide several interfaces for browsing as well as searching the stored data. A notable example of such online databases is the Rfam database [8].

Rfam database

The Rfam databases stores a collection of ncRNA multiple sequence-structure alignments grouped into families. The alignments are constructed using the Infernal software introduced earlier in section 1.4. The input to Infernal is a hand-made multiple alignment of the known members of an RNA family annotated with secondary structure information. This alignment is referred to as the seed alignment. From the seed alignment, Infernal constructs a CM profile and uses it to search for other members of the RNA family in a nucleotide sequence database. The resultant matches are then aligned to the CM, forming what is referred to as the full alignment. Both the seed and the full alignments along with the CM are stored for each family in the Rfam database [10].

1.6 Index-based search

Index-based search algorithms are sequence search algorithms that rely on constructing a special search index once and performing multiple searches over the index with a reduced per-search time complexity. Several types of indexes and index-based algorithms exist, each has its advantages and drawbacks.

The following are some of the most common index structures:

Suffix tree

The suffix tree is a data structure that gives a hierarchial representation of all the suffixes of a given string. Each path from the root to a leaf of the suffix tree corresponds to a suffix of the string represented by the tree.

Given a string S of length n and a search pattern P of length m, deciding whether P is a substring of S can be easily computed on a suffix tree of S in $O(m)$ time. Enumerating all z occurrences of P in S can be performed in $O(m + z)$ time. Notice that both times are independent of n.

There exist several algorithms for constructing the suffix tree of the search string S in $O(n)$ time [15, 23]. The space consumed by the suffix tree is also $O(n)$.

The main drawback of suffix trees is their large space consumption which, although linear, can reach up to $20n$ bytes. This drawback limits the practical applicability of suffix trees, especially in bioinformatics where search databases are usually large.

Suffix array

The suffix array is a more space-efficient index structure than the suffix tree. It stores the starting points of the suffixes of a string sorted in lexicographical order. A suffix array for a string of length n can be stored in $4n$ bytes instead of $20n$ for the suffix tree. A linear time algorithm for suffix array construction is introduced in [20]. Algorithms for solving both the decision and enumeration problems on a suffix array in the same time complexities as on a suffix tree are introduced in [3]. In addition, a generic framework for mapping any bottom-up traversal suffix tree algorithm to enhanced suffix arrays (a suffix array extended with a table) is introduced in [2].

Due to its lower space consumption, the suffix array is more practically applicable than the suffix tree. This is also supported by the existence of algorithms for solving many suffix tree problem on the suffix array.

Affix tree and affix array

The structure of the suffix tree allows using it for matching in only one direction (left to right), i.e. the search pattern can not be introduced in reverse direction. To overcome this limitation, a suffix tree is augmented with another suffix tree for the same string in reverse direction, this is known as a *reverse prefix tree*. Corresponding nodes in both tree are connected and the two trees are merged to form a single data structure known as the *affix trees*. An algorithm for constructing affix trees in $O(n)$ time is introduced in [13].

The affix array has the same functionality as the affix tree, yet more space efficient. It extends the suffix array for performing bi-directional matching in the same way the affix tree extends the suffix tree. Like affix trees affix arrays can be constructed in $O(n)$ time [22]. Affix arrays are useful in applications where bi-directional matching is required such as RNA secondary structure analysis.

AfSearch

AfSearch is an index-based search tool that uses affix arrays to search nucleotide sequence databases for given search patterns. The tool is based on ideas and concepts from [22]. Search patterns will be referred to throughout this thesis as *descriptors*.

Descriptors are composed of a sequence pattern and a secondary structure annotation. Sequence patterns can contain wildcards (i.e. positions that can match any character). However, the length of the pattern has to be

```
GC*****GGU****CC***G*GC
(((((((((.....)))))))))
```

Figure 1.2: Example descriptor

```
CCACCUUAAGAGC.GCG.UCGC.AGCCU...GGGGA.GGCUCCCGGCACCGAGACCAAUG
CCACCUUAAGAGC.GCGCUCGCCAGCCUGGGCGGAGCGGCUCCCGGCGCCGAGACCAAUG
CCACCUUAAG.GCCGCGCUCGCCAGCCUCGGCGGGGCGGCUCCCGCCGCCGCAACCAAUG
CCACCUUAAG.GCCGCGCUCGCCAGCCUCGGCGGGGCGGCUCCCGCCGCCGCAACCAAUG
........((.((.((....))..))))..(((((((((...))))).))))........
```

Figure 1.3: Example multiple alignment with secondary structure annotation

fixed, i.e. the pattern has to be solid. Secondary structures are described in dot-bracket notation, and based on the allowed base pairings, afSearch only matches sequences that can fold into the given secondary structure. AfSearch is capable of handling only hairpin structures, as defined in [27]. Figure 1.2 shows an example descriptor.

1.7 Contribution

This projects investigates the feasibility of using index-based search, in particular affix-array based search, for performing RNA homology search. The project implements the descriptor/pattern generation part of a new RNA homology search tool that employs affix array search. The new tool is compared with several other search methods in terms of efficiency, sensitivity, specificity and other measures.

The program takes as input a multiple sequence alignment of RNA with consensus secondary structure annotation. From the alignment, the program generates a set of patterns or descriptors that are used by the affix-array search tool afSearch, introduced in section 1.6, to search a nucleotide sequence database for matching RNA sequences. Figure 1.3 shows an example of a multiple alignment with secondary structure annotation.

The project introduces 2 different schemes for descriptor generation. Each scheme has a set of parameters for tuning the descriptor generation process. An internal comparison is performed to find the best scheme and the optimal parameters for descriptor generation and search.

Chapter 2

Methods

This section introduces the processes and steps through which the patterns/descriptors for the affix-array search tool afSearch are constructed. The input to the process is a multiple alignment of the sequences of an RNA family with secondary structure annotation.

Given a multiple alignment of an RNA family with secondary structure annotation, the process goes through the following 4 steps:

1. Hairpin extraction
2. Descriptor generation
3. Search
4. Chaining and filtering

hairpin extraction detects and extracts hairpin regions out of the given multiple alignment.

descriptor generation takes the hairpin regions from the hairpin extraction step and produces one or more search descriptors for every hairpin region.

search takes as input the descriptors generated by the second step and uses the afSearch affix array search tool to find RNA sequences matching the given descriptors in a nucleotide sequence database.

chaining ensures that matches from different hairpins appear in the correct order.

filtering eliminates sequences with less than the required number of correct order matches.

Detailed discussions of these steps are presented in the following sections.

2.1 Hairpin extraction

Due to the fact that the afSearch tool can only handle hairpin secondary structures (see section 1.6 page 10), the first step of the search process is extracting the hairpin columns out of the multiple alignment. This is done by examining the secondary structure annotation of the alignment. Other columns of the alignment that are not inside a hairpin region are discarded. Figure 2.1 shows an example alignment with hairpin columns highlighted.

```
CCACCUUAAGAGC.GCG.UCGC.AGCCU...GGGGA.GGCUCCCGGCACCGAGACCAAUG
CCACCUUAAGAGC.GCGCUCGCCAGCCUGGGCGGAGCGGCUCCCGGCGCCGAGACCAAUG
CCACCUUAAG.GCCGCGCUCGCCAGCCUCGGCGGGGCGGCUCCCGCCGCCGCAACCAAUG
CCACCUUAAG.GCCGCGCUCGCCAGCCUCGGCGGGGCGGCUCCCGCCGCCGCAACCAAUG
........((.((.((....))..))))..(((((((((...))))).))))........
```

Figure 2.1: Example multiple alignment with hairpin columns highlighted

Procedure

The first step in the hairpin detection procedure is matching corresponding opening and closing brackets in the secondary structure. This is done by traversing the secondary structure form left to right. When an opening bracket is encountered, its position is pushed on a stack. When a closing bracket is encountered, it is paired with the topmost element on the stack and that element is popped off the stack. If an opening bracket is encountered after one or more closing brackets, then the region between last closing bracket and its corresponding opening bracket is marked as a hairpin region and the stack is cleared. Algorithm 1 on page 15 simultaneously performs the pairing and hairpin detection procedures.

Hairpin order

The order in which the hairpins appear in the alignment is significant and is used by the chaining step explained in section 2.4. To begin discussing the order of hairpins we need to define the set of all hairpins in an alignment as follows:

Definition 11 (Hairpin set)**.** *Given an alignment A, the set of all hairpins of A, denoted as R_A is defined as follows: $R_A = \{(i, i') : 1 \leq i < i' \leq length[A]\}$ which denotes that the region in the alignment between positions i and i' inclusive is a hairpin region according to definition 5. Hairpin regions cannot overlap, i.e. $\forall_{(i,i'),(j,j') \in R_A} : (i < j \iff i' < j)$ and $(i = j \iff i' = j')$.*

```
AGAGC.GCG.UCGC.AGCCU  .GGGGA.GGCUCCCGGCACCGA
AGAGC.GCGCUCGCCAGCCU  GCGGAGCGGCUCCCGGCGCCGA
AG.GCCGCGCUCGCCAGCCU  GCGGGGCGGCUCCCGCCGCCGC
AG.GCCGCGCUCGCCAGCCU  GCGGGGCGGCUCCCGCCGCCGC
((.((.((....))..))))  ((((((((...))))).))))
```

Figure 2.2: Hairpin columns extracted from example alignment in figure 2.1. This set of columns is the input to the descriptor generation phase.

The definition above induces a total order over the hairpin regions in an alignment, thus making the hairpin set a totally ordered set.

Definition 12 (Hairpin total order)**.** *The total order of the hairpins of an alignment A is defined by the strict total order relation $(<)$ over the set R_A as follows: For $a = (i, i'), b = (j, j') \in R_A$, $a < b \iff i < j$.*

The relation defined above will be referred to throuout the thesis as "*before*".

Properties of the strict total order:
for a, b and c in R_A:

1. irreflexive: $\neg(a < a)$, i.e. A hairpin cannot be before itself.
2. asymmetric: $(a < b) \implies \neg(b < a)$, i.e. if hairpin a is before b, then b cannot be before a.
3. transitive: $(a < b) \wedge (b < c) \implies (a < c)$, i.e. if hairpin a is before b and b is before c, then a is before c.
4. trichotomous: $\neg(a < b) \wedge \neg(b < a) \implies a = b$.

Proof of strict total order properties. From definition 12, $a < b \iff i < j$, and the fact that $i, j \in \mathbb{N}$, the properties above follow directly from the strict total order of natural numbers. □

2.2 Descriptor generation

The descriptor generation step takes as input the extracted hairpin columns produced by the hairpin extraction step (see figure 2.2). Each one of those subalignments is processed in the same way producing one (or more) descriptors.

Algorithm 1 Finding hairpin regions

```
Let $D$ be the secondary structure
for $i = 1$ to $|D|$ do
  if $D_i$ = ‘(’ then
    if $last$ = ‘)’ then
      Declare $pair$ as hairpin region
      Clear stack
    end if
    Push $i$
    $last \leftarrow$ ‘(’
  else if $D_i$ = ‘)’ then
    if Stack is not empty then
      Pop $j$
      $pair \leftarrow (j, i)$
    end if
    $last \leftarrow$ ‘)’
  end if
end for
```

Definition 13 (Descriptor). *Given an alignment A over a finite alphabet Σ, such that* **‘*’** $\notin \Sigma$*, a descriptor D over the alignment is defined as follows: $D = (P, T)$, where P is a sequence over the alphabet ($\Sigma - \{\_\} \cup \{*\}$) defining the primary sequence pattern of the descriptor, and $T = \{(i, i') : 1 \leq i < i' \leq |P|\}$ with the same constraints as the definition of secondary structure (see definition 3 on page 4) defining the secondary structure constraints of the descriptor.*

In practice, the secondary structure of descriptors is expressed in dot-bracket notation as defined in definition 4 on page 5.

Since the afSearch tool can only handle solid patterns, the descriptor generation scheme must first deal with gaps or insertions, which represent differences in the lengths of the sequences in the alignment. This thesis presents two scheme for descriptor generation that differ mainly in the way they deal with gaps. The two schemes are presented in the following subsections.

2.2.1 First scheme

The first descriptor generation scheme deals with the gaps problem in a fairly simple way. The scheme simply eliminates columns with high gap content. The choice of columns to eliminate is controlled by one parameter, namely the

```
AGAGC.GCG.UCGC.AGCCU         AGGCGCG.UCGC.AGCCU
AGAGC.GCGCUCGCCAGCCU         AGGCGCGCUCGCCAGCCU
AG.GCCGCGCUCGCCAGCCU   ⇒     AGGCGCGCUCGCCAGCCU
AG.GCCGCGCUCGCCAGCCU         AGGCGCGCUCGCCAGCCU
((.((.((....))..))))         ((((((....))..))))
```

Figure 2.3: Gap removal process. On the left, columns to be eliminated are highlighted. On the right, the subalignment after gap removal. In this example, “gap ratio” = 0.6.

“gap ratio”, which specifies the maximum allowed gap:row ratio in a column. If the number of gaps exceeds the gap ratio, the column is eliminated.

Algorithm 2 below describes briefly the process of gap elimination. In the algorithm, A refers to a hairpin subalignment and *gapcount*(c) returns the number of gaps in an alignment column c. Figure 2.3 demonstrates the process.

Algorithm 2 Gap elimination algorithm

```
for all $c$ in $C_A$ do
    let $gaps \leftarrow gapcount(c)$
    if $gaps/|c| \geq$ “gap ratio” then
        Eliminate $c$
    end if
end for
```

Using the remaining columns, the number of occurrences of every nucleotide in the column is counted and an information content measure is calculated based on the counts. Two methods for calculating the information content are presented in section 2.2.3.

The information content measure is used to decide whether a position will be a wildcard or not. The choice of wildcard positions is controlled by 2 parameter, the “wildcard ratio” and the “maximum score”. The wildcard ratio determines the maximum percentage of columns to be wildcards. The maximum score determines the maximum allowed wildcard score i.e. if a column has a higher score than the wildcard score, it will never be a wildcard.

This wildcard ratio parameter is used to calculate a threshold for the minimum allowed information content. Algorithm 3 below describes briefly the descriptor generation procedure. In the algorithm, A is the subalignment with gaps eliminated, $infocontent(c)$ returns the information content of an alignment column c, *mode*(c) returns the statistical mode (i.e. the most

```
.GGGGA.GGCUCCCGGCACCGA
GCGGAGCGGCUCCCGGCGCCGA
GCGGGGCGGCUCCCGCCGCCGC
GCGGGGCGGCUCCCGCCGCCGC
G*GG**CGGCUCCCG*C*CCG*
```

Figure 2.4: Descriptor generation. The last line is the generated descriptor.

abundant element) of a column. The '*' symbol denotes a wildcard. Figure 2.4 shows a subalignment with the generated descriptor highlighted below.

Algorithm 3 Descriptor generation algorithm

```
for $i = 1$ to $|C_A|$ do
    $info[i] = infocontent(C_{A,i})$
end for
Sort $info$
$index \leftarrow \lfloor$ "wildcard ratio" $\cdot |info| \rfloor$
$R \leftarrow info[index]$
for $i = 1$ to $|C_A|$ do
    if $infocontent(C_{A,i}) \geq R$ then
        $descriptor[i] = mode(C_{A,i})$
    else
        $descriptor[i] =$ '*'
    end if
end for
```

If the information content of a column is lower than the calculated threshold and the given maximum score, a wilcard is used to represent the column. Otherwise the most abundant nucleotide or "character" is used to represent the column. The generated pattern is combined with the secondary structure notation of the hairpin to produce a descriptor such as the one shown in figure 1.2 on page 11.

2.2.2 Second scheme

The first scheme produces only one descriptor per hairpin. This limits the matches from such a descriptor to hairpins of only one length. Thus if there large variations in the lengths/gaps of the sequences in the alignment, the generated descriptor would only express a few of them. To better understand the problem, consider the alignment in figure 2.5. The strong variations in the lengths and gap patterns of the sequences will cause any single descriptor

```
GCUCAAUC..GGU...AGAGC
GUUUAAU...GGA...AAAAC
GUGAAAAA..GGUA..AACAC
ACUGAGU...GGUUU.AAGGU
GCUGAA....GUU...UAGGC
GUUUAAU...GA....AGAAU
GUGAAAU...GGUA..GACAC
GUUUAA....UC....AAAAC
```

Figure 2.5: A multiple alignment with highly varying gap patterns.

generated for the alignment using the first scheme to match only one of the sequences since although the sequences share many gap positions, no to sequences have the same gap pattern.

The second descriptor generation solves this problem by generating multiple descriptors for a single hairpin. Each descriptor is generated from a group of sequences with identical gap patterns. This is done by grouping the sequences of a certain hairpin loop by gap patterns, giving groups with more sequences a higher priority.

Definition 14 (Gap pattern)**.** *The gap pattern of a sequence S is defined as the set of all indices/positions in the sequence that are gaps:*
$G_S = \{i : 1 \leq i \leq |S| \text{ and } S_i = \text{`\_'}\}$.

Grouping procedure

The grouping procedure takes the hairpin subalignment and performs the following steps iteratively until it ends with an empty set:

1. remove a sequence at random from the alignment.
2. check all sequences in the alignment, if a sequence has the same gap pattern as the sequence taken in step 1, do the following:
   (a) remove the sequence from the alignment.
   (b) put it in the same group as the sequence taken in step 1.

Algorithm 4 below explains the procedure in pseudocode. In the algorithm, A denotes the processed alignment and G_s is the gap-pattern of a string s, as defined in definition 14 on page 18. At the end of the algorithm, the variable *groups* will contain the sequences of the alignment divided into groups by gap patterns.

Algorithm 4 Sequence grouping according to gap patterns

```
$groups \leftarrow \emptyset$
while $A$ is not empty do
  $s \leftarrow$ random sequence from $A$
  remove $s$ from $A$
  $group \leftarrow \{s\}$
  for all $s'$ in $A$ do
    if $G_{s'} = G_s$ then
      $group \leftarrow group \cup \{s'\}$
      remove $s'$ from $A$
    end if
  end for
  $groups \leftarrow groups \cup \{group\}$
end while
```

Prioritisation

Instead of the "gap ratio" parameter used for the first scheme, the second scheme is controlled by 2 other parameters The "max groups" parameter controls the maximum number of groups that will be used to generate descriptors. The "descriptor ratio" parameter governs the total number of sequences in the groups that will be used to generate descriptors, to the total number of sequences in the alignment. Algorithm 5 below clarifies in pseudocode the prioritisation process. In the code, *descgroups* is the set of groups that will be used to generate descriptors, such that $descgroups \subseteq groups$.

Algorithm 5 Prioritization

```
  $union \leftarrow \emptyset$
  for all $group$ in $descgroups$ do
    $union \leftarrow union \cup group$
  end for
Ensure: $|union|/|A| \leq$ "descriptor ratio"
Ensure: $|descgroups| \leq$ "max groups"
```

To choose which groups should be used, the second scheme employs a simple prioritisation technique, giving a higher priority to groups with a larger number of sequences. In the implementation, this is achieved by sorting the groups in descending order by size and taking the first n groups such that $list_{1n}$ satisfied the above conditions while $list_{1(n+1)}$ violates one or more of the conditions.

Descriptor generation

After the groups are decided, the common gap columns in each group are eliminated and the descriptors are generated from the subalignment as in the first scheme. The consensus computation is controlled by the same 2 parameters used for the the first scheme, namely, “wildcard ratio” and “maximum score”.

Algorithm 6 Second scheme descriptor generation

```
for all group in descgroups do
    eliminate gaps from group
    generate descriptor from group
end for
```

2.2.3 Information content calculation

Two different methods were used for calculating the information content of a column used by by the first and second descriptor generation schemes described above.

The first method is straight-forward, it computes the percentage of the most abundant element from the total number of columns. Thus if we define $N(x, c)$ as the number of occurrences of element x in column c, and $Max(c)$ as the most abundant element in column c, then the information content of a column c would be:

$$\frac{N(Max(c), c)}{|c|}$$

The second information content calculation method uses the LOGOS measure explained in [19]. For a column c, the uncertainty measure $H(c)$ is defined as follows:

$$H(c) = \sum_{b=\text{‘A’}}^{\text{‘U’}} f(b, c) \log_2 f(b, c)$$

where b goes through the four bases $\{A, G, C, U\}$, and $f(b, c)$ is the frequency of base b in column c. The information content of a column c is then defined by the following formula:

$$R(c) = 2 - (H(c) + e(n))$$

where $e(n)$ is a correction factor.

The results from both schemes are compared in the results chapter.

2.2.4 General definitions

The following are some descriptor-related definitions that are independent of the descriptor generation scheme used.

The following is the definition of the function that maps descriptors to the hairpin from which they were generated.

Definition 15 (Source hairpin)**.** *Given an alignment A, and using M_A to refer to the set of all descriptors from A, we define a function $Hp : M_A \longrightarrow R_A$ as follows: $Hp(D) =$ the hairpin from which descriptor D was generated.*

The function defined above will be referred to as the *Source hairpin* function.

Like hairpins, descriptors are also ordered. However since one hairpin can generate multiple descriptors, the order of descriptors is a partial order.

Definition 16 (Descriptor partial order)**.** *Given some alignment A and two descriptors a and b in the set of all alignment descriptors M_A, the strict partial order relation $(<)$ over M_A is defined as follows: $a < b \iff Hp(a) < Hp(b)$.*

This relation will sometimes be referred to as "*before*".

A descriptor is thus regarded as "before" another descriptor if the former is generated from a hairpin that appears earlier in the alignment than the the one from which the latter is generated. Note that in the second descriptor generation scheme, more than one descriptor can be generated from one hairpin, thus the descriptors can be regarded as overlapping.

2.3 Search

After the descriptors are generated for every hairpin in the alignment, the descriptors are used by the afSearch tool to search through a nucleotide sequence database for sequences matching one or more of the given descriptors. Along with the matched sequences, the tool also returns the matching positions in the sequence and the input descriptor that produced the match. This information is essential for the chaining step explained in section 2.4.

To define matching, the notion of *allowed* base pairs has to be defined first. The following is the definition of the set of possible base pairs:

Definition 17 (Possible base pairings)**.** *The set of possible base pairings Y over the RNA alphabet $\Sigma = \{A, G, C, U\}$, is defined as follows: $Y = (A, B)|A, B \in \Sigma$. For all pairs $(A, B) \in Y$, A can pair with B.*

The matching process is defined as follows:

Definition 18 (Matching)**.** *Given a descriptor $D = (P, T)$ and sequence S and a set of possible base pairings Y, such that $\Sigma_P = \Sigma_S \cup \{*\}$, D* matches *S at position p if all of the following is true:*

1. *$P_j =$ '`*`' or $S_{p+j-1} = P_j$, for $1 \leq j \leq |P|$.*
2. *$(S_j, S_{j'}) \in Y or (S_{j'}, S_j) \in Y$, for all $(j, j') \in T$. i.e. S_j can pair with $S_{j'}$.*

A match *can be defined as the triple (S, D, p). The search process returns a set of matches.*

In the definition above, condition 1 means that the substring $S_{p(p+|P|-1)}$ matches the pattern P. Condition 2 means that the matched substring can make the base pairings of the secondary structure of the descriptor.

Figure 2.6 shows an example descriptor and figure 2.7 shows a sequence with the matching region of the descriptor in figure 2.6 highlighted. Notice that the descriptor matches at position $i = 11$ on the sequence in figure 2.7.

```
C**C*CCC*A*CG*G**G
((((((......))))))
```

Figure 2.6: A descriptor. Above is the primary sequence pattern. Below is the secondary structure in dot-bracket notation.

```
CGGCGGGACACCCCUCCCCAACGAGGGGCUUCUAUCUGAGAGGAUAGCAUAU
```

Figure 2.7: A sequence with the matching region of the descriptor in figure 2.6 highlighted.

Note that several descriptor can match the same sequence and even the same descriptor can match a single sequence several times at different positions.

Sequence match set

The set of matches returned by search process contains matches from multiple sequences in the nucleotide database. It is useful however, to define the set of matches in a specific sequence.

Definition 19 (Sequence match set)**.** *Given a sequence K in the nucleotide sequence database used by the search process, the set of all matches in K is defined as follows: $M_K = \{M = (S, D, p) : M$ is a match and $S = K\}$.*

Match order

The match order is defined on matches in the same sequence, i.e. the notion of match order only applies to sequence match sets defined in definition 19. Unlike hairpins, different matches matches may overlap, thus the definition of match order will be different from that of hairpins.

Definition 20 (Match partial order)**.** *The partial order of the matches of a sequence S, is defined by the strict partial order relation $(<)$ over the set M_S as follows:*

For all $m_1 = (S, D_1, p_1), m_2 = (S, D_2, p_2) \in M_S$, $m_1 < m_2 \iff e_1 < p_2$, where $e_i = p_i + |D_i|$ denotes the ending point of a match m_i.

From the definition above, match is $<$ another match if the former ends before the beginning of the latter. The set M_S along with the $(<)$ operator defined above form a strict partially ordered set.

The definition above formalizes the intuitive notion of the order of matches.

2.4 Chaining and filtering

Recall that every RNA alignment needs to have one or more hairpins. From every hairpin, one or more descriptors are generated. The descriptors are given to the search engine which uses them to search a nucleotide sequence database and find the matching positions of each descriptor. Recall also that several matches can exist in a single sequence. To make sure that the matches actually identify a correct member of the family and not just matched by coincidence, the chaining process makes sure that the matches appear in the correct order and find the longest sequence of matches that is in the correct order.

Depending on the number of hairpins in the alignment of the RNA family to be searched, a search sequence is required to have a proportional number of matches. In addition, the matches must not overlap and must appear in the same order of their corresponding hairpins (and hence descriptors). This leads to the problem of chaining matches to find the longest sequence of correctly ordered matches in some sequence. To further discuss chaining, a second type of partial order on sequence match sets is defined as follows.

Definition 21 (Chainability)**.** *Given any 2 matches $m_1 = (S, D_1, p_1)$ and $m_2 = (S, D_2, p_2)$ in the match set M_S of some sequence S, we define the chainability relation on the set M_S as follows: $m1$ is chainable to $m2$ if-and-only-if $m_1 < m_2$ as defined in defintion 20 and $D_1 < D_2$ as defined in definition 16.*

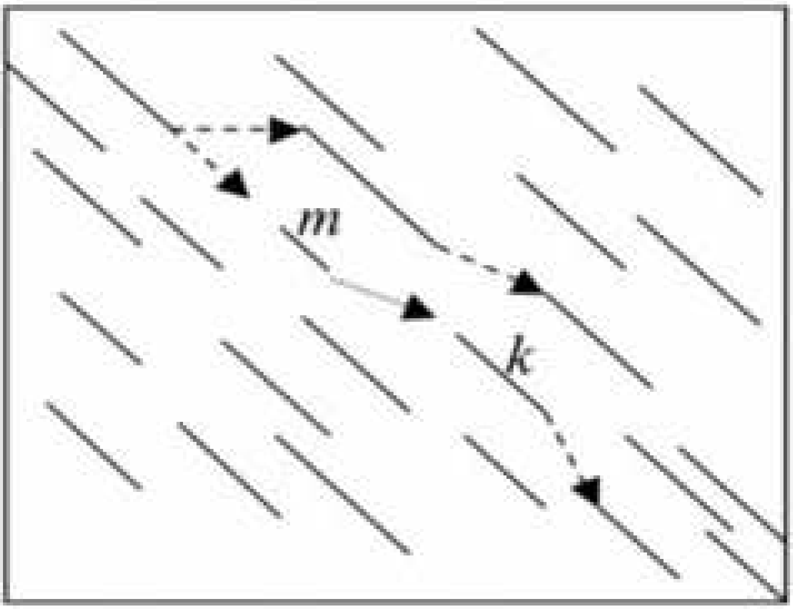


Figure 2.8: The chaining process. Figure taken from [25]

The relation will be referred to as "*chainable to*".

From the definition above, a match is chainable to another match if the former ends before the beginning of the latter and if the former comes from a descriptor and hence a hairpin that is before that of the latter.

This definition induces as second type of strict partial order on the match set of a sequence. Using the definition above, chaining can be defined as the longest sequence of chainable matches in a the match set of a sequence. Figure 2.8 is visual demonstration of the chaining process. In the figure, the horizontal axis represents a sequence to be chained, the vertical axis represents the descriptors and the diagonal lines represent matches between regions of the descriptors and the sequence. An arrow represents chainability.

Due to the fact that every strict partial order corresponds to a directed acyclic graph (DAG), one can model the matches in the match set as nodes in the graph and chainability relations as edges. Using this model, chaining is simply finding the longest path (or critical path) in the graph.

In the implementation, chaining is done using the graph approach described in the previous paragraph. For a sequence with m matches, this implementation of chaining runs in $O(m^2)$. This is due to the fact that the number of edges in the graph is bounded from above by $m(m-1)/2$ so constructing the graph requires $O(m^2)$ time and the longest (or critical) path algorithm runs on DAGs in $O(m)$ time giving a total of $O(m^2)$. This approach is taken from [11].

Filtering

The chaining step takes the match set of some sequence as input and return the length of the maximum possible chain of matches and possibly the chain itself. If a sequence does not satisfy a minimum chain length, which is proportional to the number of hairpins extracted from the alignment, the sequence is eliminated from the result set. This is referred to as *filtering*.

2.5 Summary

Figure 2.9 summarises the overall descriptor generation and search process. The input to the process is a multiple sequence alignment with secondary structure annotation. The process goes through the steps labeled in the diagram as follows:

1. Hairpin regions are extracted from the alignment.

2. Depending on the descriptor generation scheme used, gaps are eliminated and descriptors are generated for every hairpin subalignment.

3. The descriptors are sent to the affix-array based search tool afSearch, which searches through a nucleotide sequence index and returns sequences with one or more matches.

4. The matched sequences are sent to the chaining process, which finds the longest possible chain of matches for every sequence.

5. Depending on the number of hairpins in the alignment, sequences with chaining results lower than some threshold are removed from the result set and the search results are returned.

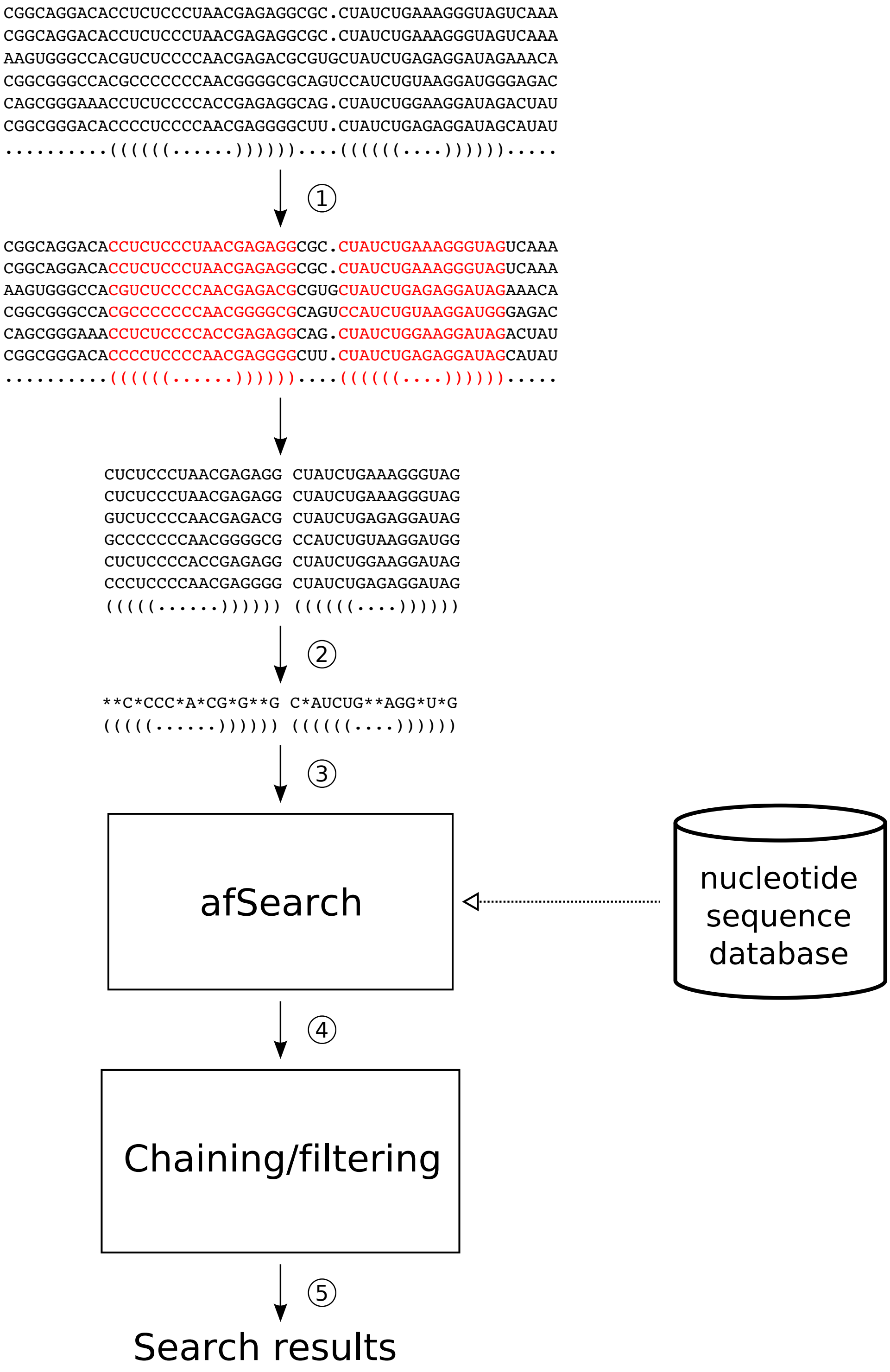

CGGCAGGACACCUCUCCCUAACGAGAGGCGC.CUAUCUGAAAGGGUAGUCAAA
CGGCAGGACACCUCUCCCUAACGAGAGGCGC.CUAUCUGAAAGGGUAGUCAAA
AAGUGGGCCACGUCUCCCCAACGAGACGCGUGCUAUCUGAGAGGAUAGAAACA
CGGCGGGCCACGCCCCCCCAACGGGGCGCAGUCCAUCUGUAAGGAUGGGAGAC
CAGCGGGAAACCUCUCCCCACCGAGAGGCAG.CUAUCUGGAAGGAUAGACUAU
CGGCGGGACACCCCUCCCCAACGAGGGGCUU.CUAUCUGAGAGGAUAGCAUAU
..........((((((......))))))....((((((....)))))).....
①
CGGCAGGACACCUCUCCCUAACGAGAGGCGC.CUAUCUGAAAGGGUAGUCAAA
CGGCAGGACACCUCUCCCUAACGAGAGGCGC.CUAUCUGAAAGGGUAGUCAAA
AAGUGGGCCACGUCUCCCCAACGAGACGCGUGCUAUCUGAGAGGAUAGAAACA
CGGCGGGCCACGCCCCCCCAACGGGGCGCAGUCCAUCUGUAAGGAUGGGAGAC
CAGCGGGAAACCUCUCCCCACCGAGAGGCAG.CUAUCUGGAAGGAUAGACUAU
CGGCGGGACACCCCUCCCCAACGAGGGGCUU.CUAUCUGAGAGGAUAGCAUAU
..........((((((......))))))....((((((....)))))).....
CUCUCCCUAACGAGAGG CUAUCUGAAAGGGUAG
CUCUCCCUAACGAGAGG CUAUCUGAAAGGGUAG
GUCUCCCCAACGAGACG CUAUCUGAGAGGAUAG
GCCCCCCCAACGGGGCG CCAUCUGUAAGGAUGG
CUCUCCCCACCGAGAGG CUAUCUGGAAGGAUAG
CCCUCCCCAACGAGGGG CUAUCUGAGAGGAUAG
((((((......)))))) ((((((....))))))
②
C*CCC*A*CG*GG C*AUCUG**AGG*U*G
((((((......)))))) ((((((....))))))
③
afSearch
nucleotide sequence database
④
Chaining/filtering
⑤
Search results

Figure 2.9: Process overview

Chapter 3

Evaluation and Results

3.1 Evaluation scheme

The Rfam database (section 1.5, page 9) is an ideal resource to use for evaluating the effectiveness of the descriptor generation methods described in the previous section, and comparing the different schemes and parameter settings.

To search a certain RNA family, the hand-curated Rfam seed alignment of the family is used as input to train the descriptors. The descriptors are then used by the afSearch tool and the search results are compared to the sequences in the full alignment taken from the Rfam database. As explained is section 1.5, the full alignment is produced using the Infernal tool.

The searches are performed using version 8.1 of the Rfam database. This version contains the alignments of 630 RNA families. For every test, the procedures described in chapter 2 are repeated on all the families.

Every search cycles traverses through all the families of the database performing the following steps:

1. generate descriptors from the seed alignment
2. perform search using afSearch
3. compare search results with sequences in full alignment
4. compute statistical measures

The rest of this chapter presents the results and comparisons of the methods described earlier.

3.2 Terminology

Throughout this thesis, a search match will be denoted as *correct* if it exists in the full alignment of the searched RNA family.

In the context of a nucleotide search, any sequence in the nucleotide sequence database can be described using one of the following four terms:

True positive is a sequence that is matched by the search and is in the full alignment of the searched family.

False positive is a sequence that is matched by the search but is not in the full alignment of the searched family.

True negative is a sequence that is not matched by the search and is not in the full alignment of the searched family.

False negative is a sequence that is not matched by the search but is in the full alignment of the searched family.

3.3 Statistical measures

This section defines some statistical measures that are used to evaluate and compare the search results.

The following are the definitions of three important statistical measures:

Definition 22 (Sensitivity)**.**

$$\text{sensitivity} = \frac{\text{number of True Positives}}{\text{number of True Positives} + \text{number of False Negatives}}$$

Definition 23 (Specificity)**.**

$$\text{specificity} = \frac{\text{number of True Negatives}}{\text{number of True Negatives} + \text{number of False Positives}}$$

Definition 24 (Positive predictive value (PPV))**.**

$$PPV = \frac{\text{number of True Positives}}{\text{number of True Positives} + \text{number of False Positives}}$$

Definition 25 (Ideal family)**.** *In the context of a nucleotide search, an* ideal family *is one for which* sensitivity = specificity = 100%*.*

From the definition above, an ideal family is one for which all correct sequences are matched and no false positives are matched.

3.4 Results

All the tests described in this section were run on a 2.13 GHz Intel Core2 machine with 3GB of memory. The pattern that is used to perform the tests is to fix all of the parameters for search/descriptor generation and to alter only one at a time, observing effects on the variables and statistical measures defined in the previous two sections. In all of the results, search time denotes the time taken to search all 630 families of Rfam 8.1.

3.4.1 First scheme

This section presents the results of searches performed using the first descriptor generation scheme.

Match ratio

In the following series of tests, all the parameters are fixed except for the match ratio parameter. The table below shows the values to which the parameters are fixed.

Parameter	Value
Descriptor scheme	scheme 1
Chaining ratio	0.5
Gap ratio	0.7
Max score	0.95
Information content	conservation ratio

The following table shows the effect of changing the match ratio on the statistical measures and the search time.

Match Ratio	Avg. Sensitivity	Avg. PPV	Ideal family count	search time (s)
0.00	0.0436	0.97289	25	10.36
0.20	0.19622	0.978	42	30.29
0.40	0.3598	0.97006	49	127.89
0.60	0.42569	0.94929	52	401.97
0.70	0.46929	0.93128	52	710.57
0.80	0.50998	0.9153	52	961.54

The table above shows that increasing the wildcard ratio, and hence the number of wildcards in a descriptor, increases the average sensitivity, ideal family count and search time while decreasing the average PPV.

Information content

In this section, all the parameters are fixed except for the information content which uses the LOGOS method instead of the conservation ratio. The results are compared to those from the previous series of tests. The following are the fixed parameters.

Parameter	Value
Descriptor scheme	scheme 1
Chaining ratio	0.5
Gap ratio	0.7
Max score	1.90
Information content	LOGOS

The following table shows the results of using the LOGOS method instead of the conservation ratio to decide which positions in a descriptor should be wildcards.

Match Ratio	Avg. Sensitivity	Avg. PPV	Ideal family count	search time (s)
0.40	0.36108	0.96921	49	140.16
0.60	0.42997	0.95042	52	487.10
0.70	0.46634	0.93082	52	690.10

Comparing the table above with the results table on page 29, it can be generally noted that LOGOS produces a slightly higher average sensitivity than conservation ratio but in a trade-off for a slightly lower average PPV. There is no difference in the ideal family count.

Chaining ratio

In this section, all the parameters are fixed except for the chaining ratio, which determines the minimum number of chained matches needed to accept a search result. It is determined as a percentage of the number of hairpins in the input alignment of the RNA family. The following are the fixed parameters and their corresponding values:

Parameter	Value
Descriptor scheme	scheme 1
Gap ratio	0.7
Max score	0.95
Wildcard ratio	0.4
Information content	conservation ratio

The following table shows a comparison of results with different chaining ratios.

Chaining Ratio	Avg. Sensitivity	Avg. PPV	Ideal family count	search time (s)
0.20	0.46249	0.91225	59	140.813869
0.40	0.37324	0.96921	51	142.899285
0.50	0.36108	0.96921	49	140.161394
0.60	0.26399	0.98839	29	139.90015

From the table above, increasing the chaining ratio decreases the average sensitivity and ideal family count while increasing the average PPV. Differences in search times are negligible.

3.4.2 Second scheme

This section presents the results of searches performed using the second descriptor generation scheme.

Wildcard ratio

This section investigates the effect of changing the wildcard ratio on the results of searches performed using the second descriptor generation scheme. The following parameters are fixed:

Parameter	Value
Descriptor scheme	scheme 2
Descriptor ratio	0.7
Max score	0.95
Wildcard ratio	0.4
Information content	conservation ratio

Match Ratio	Avg. Sensitivity	Avg. PPV	Ideal family count	search time (s)
0.40	0.41883	0.97622	75	115.056093
0.60	0.52477	0.95707	79	233.492931
0.70	0.59369	0.948	79	346.513616
0.80	0.73708	0.93522	80	539.320117

Observe from the table above that although the second scheme produces, on average, more than one descriptor per hairpin loop, the search times for the second scheme are significantly lower than their corresponding search time in the first scheme. This is due to the fact that the smaller number of sequences used to generate a descriptor in the second scheme, has less variance among the bases in a column and hence less wildcards in the descriptor which decreases the search time.

Chapter 4

Conclusion

In conclusion, this new technique for RNA homology search can be seen as a trade-off between speed and accuracy or efficiency and effectiveness, that is tending more towards the efficiency side than the effectiveness side. The results in the previous chapter show that the second descriptor generation scheme outperforms the first scheme in terms of both speed and accuracy. The choice of parameters depends on the constraints of the application.

The program can thus provide a fast and reliable search tool for a subset of the families, namely the families with high sensitivity and PPV results.

Appendix A

Extensions and Implementation

This appendix introduces two extensions that were implemented for the affix-array search tool afSearch.

A.1 Multiple search extension

The afSearch tool was limited to performing only a single search per run, i.e. the program starts and loads the index, performs one search using the given descriptors and then exits. This pattern causes a lot of overhead due to the considerable amount of time needed to load the search index every time a search is performed. To overcome this limitation and to enable fast database-wide searching, an extension for afSearch was written to enable afSearch to load a search index once and use it to perform multiple searches thus eliminating a massive amount of overhead that would have been needed if the index were loaded to search every one of the 630 families in version 8.1 of the Rfam database.

A.2 Chaining extension

The afSearch program used an external tool called Chain2dim to perform the chaining step described in section 2.4. Using Chain2dim for this application had two drawbacks. First, it required that all potential matches (matches before chaining and filtering) be written on files on the hard drive. This process required a massive amount of time causing huge overhead for the search. The second drawback was that Chain2dim did not support multiple descriptors per hairpin, which is needed for the second descriptor generation scheme.

These two drawbacks were the motivation to implement a built-in chaining extension for the afSearch tool. The new extension extracted the search results directly from the memory, thus eliminating the overhead of writing potential matches to hard drive. The extensions also added chaining support for multiple descriptors per hairpin loop.

The extensions uses the graph approach described in section 2.4 to perform the chaining. The only disadvantage of such implementation is that it runs in $O(n^2)$ instead of $O(n \lg n)$ for Chain2dim. However due to the small size of the problem and the large overheads of Chain2dim, the extension runs a lot faster than Chain2dim.

A.3 Implementation

The descriptor generation software was implemented in Python[1], a very-high-level object oriented programming language.

The extensions described above were implemented in C.

Bibliography

[1] Python Programming Language. http://www.python.org/.

[2] M. Abouelhoda, S. Kurtz, and E. Ohlebusch. The enhanced suffix array and its applications to genome analysis. *Lecture Notes in Computer Science*, pages 449–463, 2002.

[3] M. Abouelhoda, E. Ohlebusch, and S. Kurtz. Optimal exact string matching based on suffix arrays. *Lecture Notes in Computer Science*, pages 31–43, 2002.

[4] V. Ambros. microRNAs tiny regulators with great potential. *Cell*, 107(7):823–826, 2001.

[5] S. Bernhart, I. Hofacker, S. Will, A. Gruber, and P. Stadler. RNAalifold: improved consensus structure prediction for RNA alignments. *BMC bioinformatics*, 9(1):474, 2008.

[6] S. Eddy and R. Durbin. RNA sequence analysis using covariance models. *Nucleic Acids Research*, 22(11):2079, 1994.

[7] D. N. Frank and N. R. Pace. Ribonuclease p: Unity and diversity in a trna processing ribozyme. *Annual Review of Biochemistry*, 67(1):153–180, 1998.

[8] P. Gardner, J. Daub, J. Tate, E. Nawrocki, D. Kolbe, S. Lindgreen, A. Wilkinson, R. Finn, S. Griffiths-Jones, S. Eddy, et al. Rfam: updates to the RNA families database. *Nucleic Acids Research*, 2008.

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

[10] S. Griffiths-Jones, A. Bateman, M. Marshall, A. Khanna, and S. Eddy. Rfam: An RNA family database. *Nucleic Acids Research*, 31(1):439, 2003.

[11] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[12] S. Heyne. Pairwise Comparison of RNA Secondary Structures via Exact Pattern Matches. Master's thesis, Friedrich-Schiller-Universität Jena, 2007.

[13] M. Maaß. Linear bidirectional on-line construction of affix trees. *Algorithmica*, 37(1):43–74, 2008.

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

[15] E. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM (JACM)*, 23(2):262–272, 1976.

[16] E. Nawrocki, D. Kolbe, and S. Eddy. Infernal 1.0: inference of RNA alignments. *Bioinformatics*, 25(10):1335, 2009.

[17] G. Pavesi, G. Mauri, M. Stefani, and G. Pesole. RNAProfile: an algorithm for finding conserved secondary structure motifs in unaligned RNA sequences. *Nucleic Acids Research*, 32(10):3258, 2004.

[18] D. Sankoff. Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM Journal on Applied Mathematics*, pages 810–825, 1985.

[19] T. Schneider and R. Stephens. Sequence logos: a new way to display consensus sequences. *Nucleic Acids Research*, 18(20):6097–6100, 1990.

[20] K. Schurmann and J. Stoye. An incomplex algorithm for fast suffix array construction. *Software: Practice and Experience*, 37(3):309–329, 2007.

[21] G. Storz. An Expanding Universe of Noncoding RNAs. *Science*, 296(5571):1260–1263, 2002.

[22] D. Strothmann. The affix array data structure and its applications to RNA secondary structure analysis. *Theoretical Computer Science*, 2007.

[23] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[24] G. Varani and W. McClain. The G$\cdot$ U wobble base pair: a fundamental building block of RNA structure crucial to RNA function in diverse biological systems. *EMBO reports*, 1(1):18, 2000.

[25] M. Veeramalai, Y. Ye, and A. Godzik. Tops++fatcat: Fast flexible structural alignment using constraints derived from tops+ strings model. *BMC Bioinformatics*, 9(1):358, 2008.

[26] S. Will, K. Reiche, I. Hofacker, P. Stadler, and R. Backofen. Inferring noncoding RNA families and classes by means of genome-scale structure-based clustering. *PLoS Comput. Biol*, 3(4):65, 2007.

[27] M. Zuker. Optimal computer folding of large RNA molecules using thermodynamics and oscilliary information. *Nucleic Acids Res*, 9:133–148, 1981.