Finite automata can detect microsatellites effectively in DNA. "Effectively" includes the ability to fine-tune the detection process so that redundant data is avoided, and relevant data is not missed during search.

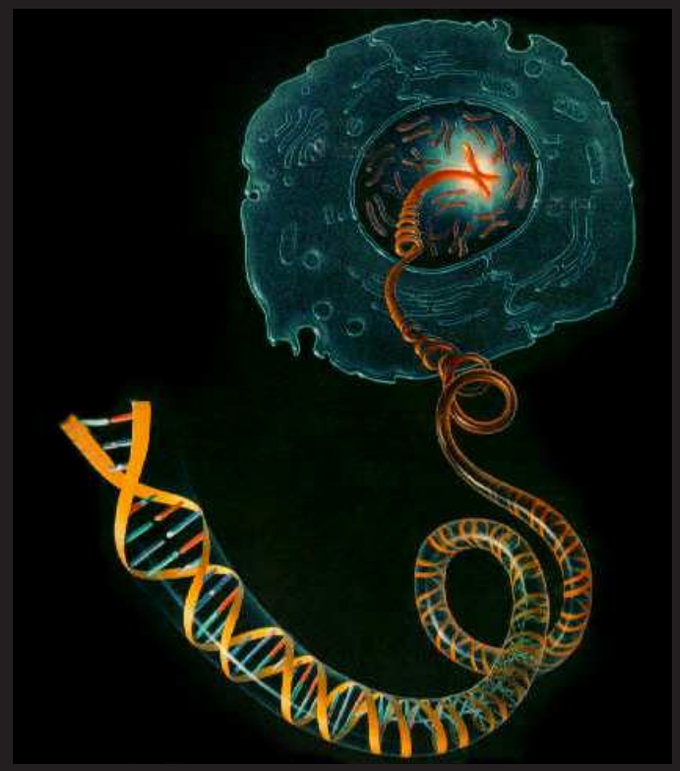
In order to verify whether the hypothesis holds, three theoretical related algorithms have been proposed based on theorems from finite automaton theory. They are generically referred to as the FireμSat algorithms. These algorithms have been implemented, and the performance of FireμSat$_2$ has been investigated and compared to other software packages. From the results obtained, it is clear that the performance of these algorithms differ in terms of attributes such as speed, memory consumption and extensibility. In respect of speed performance, FireμSat outperformed rival software packages.

FA algorithms detecting microsatellites

C de Ridder

L

Flexible finite automata-based algorithms for detecting microsatellites in DNA



Corné de Ridder

Latex Press

Flexible finite automata-based algorithms for detecting microsatellites in DNA

Corné de Ridder
Department of Computer Science
University of Pretoria
Pretoria 0002
South Africa

July 24, 2010

Wer nicht vorwärts geht, der kommt zurücke.
If you're not going forward, you're going backward.

— J.W. von Goethe

Abstract

Apart from contributing to Computer Science, this research also contributes to Bioinformatics, a subset of the subject discipline Computational Biology. The main focus of this dissertation is the development of a data-analytical and theoretical algorithm to contribute to the analysis of DNA, and in particular, to detect microsatellites.

Microsatellites, considered in the context of this dissertation, refer to consecutive patterns contained by genomic sequences. A perfect tandem repeat is defined as a string of nucleotides which is repeated at least twice in a sequence. An approximate tandem repeat is a string of nucleotides repeated consecutively at least twice, with small differences between the instances.

The research presented in this dissertation was inspired by molecular biologists who were discovered to be visually scanning genetic sequences in search of short approximate tandem repeats or so called microsatellites.

The aim of this dissertation is to present three algorithms that search for short approximate tandem repeats. The algorithms comprise the implementation of finite automata.

Thus the hypothesis posed is as follows: *Finite automata can detect microsatellites effectively in DNA.* "Effectively" includes the ability to fine-tune the detection process so that redundant data is avoided, and relevant data is not missed during search.

In order to verify whether the hypothesis holds, three theoretical related algorithms have been proposed based on theorems from finite automaton theory. They are generically referred to as the FireμSat algorithms. These algorithms have been implemented, and the performance of FireμSat$_2$ has been investigated and compared to other software packages. From the results obtained, it is clear that the performance of these algorithms differ in terms of attributes such as speed, memory consumption and extensibility. In respect of speed performance, FireμSat outperformed rival software packages.

It will be seen that the FireμSat algorithms have several parameters that can be used to tune their search. It should be emphasized that these parameters have been devised in consultation with the intended user community, in order to enhance the usability of the software. It was found that the parameters of FireμSat can be set to detect more tandem repeats than rival software packages, but also tuned to limit the number of detected tandem repeats.

Acknowledgements

Many people have been influential in my life and in my work and I would not be who I am today without their loving support, my thanks to them for their guidance and help. A special thanks to my supervisors, Prof Kourie you are somebody who besides from being an academic supervisor can be considered a supervisor of life — thank you for all the understanding, patience, support and advice during the journey. A great thanks to my husband, Pieter, who supported me in many ways while I was writing this dissertation.

To my sons Franco and Gustav.

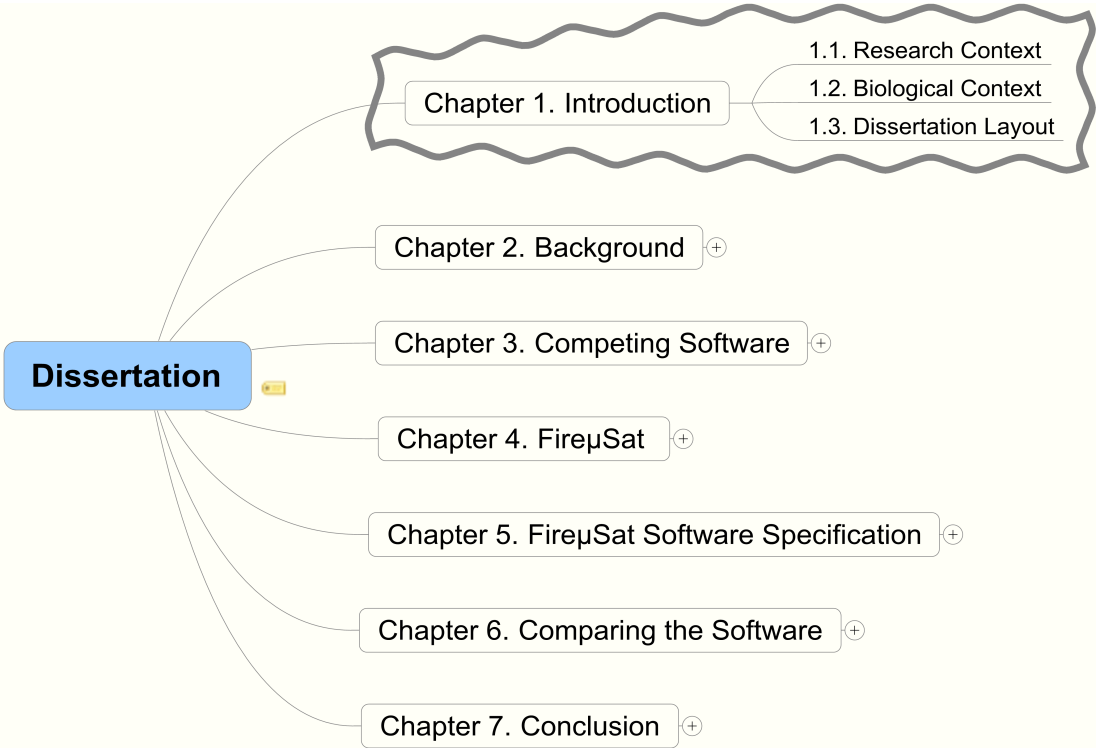
Contents

List of Tables

List of Figures

Chapter 1

Introduction



Chapter 1 starts in Section 1.1 with a discussion of the research approach followed during the research presented in this dissertation. In Section 1.2 background to the biological issues that gave rise to the research, is discussed. The chapter ends with Section 1.3 which motivates the remainder of the dissertation lay out.

1.1 Research Context

Glas *et al.* (2004) distinguish between three disciplines within Computing. These disciplines are Computer Science (CS), Software Engineering (SE) and Information Systems (IS). It is mentioned that CS examines topics related to computer concepts at technical levels of analysis by formulating processes *or* methods *or* algorithms by using mathematically-based conceptual analysis. CS does usually not rely on reference disciplines[1] [Glas *et al.* (2004)].

Research conducted by Glas *et al.* (2004) shows that the research approach adopted by 79,1% of computer scientists is formulative; and 89,33% of research conducted by computer scientists is self-referenced and self-reflective. The proposed research aims to contribute to the Computer Science knowledge framework, it is self-reflective — it relies on theorems and definitions from the discipline of Computer Theory. The conducted research is formulative in the sense that three algorithms have been formulated. Thus the research approach followed in this dissertation belongs typically to the research approach, followed most within the Computer Science domain.

Details pertaining to these algorithms are to be found in Chapter 4. The proposed algorithms have been implemented. Therefore, it is possible to measure the effectiveness of the algorithms in a quantitative manner. Quantitative research can be either experimental or quasi experimental [Goubil-Gambrell (1991)]. The conducted research is experimental in the sense that it includes the manipulation of variables and obtains various empirical[2] data that are compared to data generated by similar algorithms. Chapter 3 reports on prominent, selected software with similar objectives. Chapter 6 provides a comparison between the runtime and data generated by different software packages that search for microsatellites.

Put more generally, this study adopts a positivistic stance. Positivists prefer research methods that start with precise theories from which verifiable hypothesis can be extracted and tested in isolation [Easterbrook *et al.* (2007)]. This research has been conducted in a deductive manner. Deductive reasoning proceeds from the more general to the more specific. Thus from theory to hypothesis to observation and hopefully to confirmation [Trochim (2006)]. The following research methodology has been followed:

- *Theory:* Theoretical algorithms have been proposed based on theorems[3]

[1] Reference disciplines within the context of research analysis refers to other disciplines whose theories formed a basis for the research. Within this context self-reference indicates reference to theories or papers in the discipline under examination. Conducted research has shown that economics is for example a reference discipline of IS [Glas *et al.* (2004)].

[2] Empiricism advocates the idea that observation and measurement is the core of the scientific endeavour [Trochim (2006)].

[3] The applicable theorems are discussed in detail in Section 4.2.

from the field of Computer Theory[4].

- *Hypothesis:* Finite automata (FAs)[5] can detect microsatellites[6] effectively[7] in deoxyribonucleic acid (DNA).
- *Observation:* As mentioned the suggested algorithms have been implemented and consequently, the data generated by them, as well as their runtime, have been observed and reported on. These algorithms have been implemented in high level programming languages. The implementation details can be found in Chapter 5.
- *Confirmation:* From the observations in Chapter 5, reported on and compared to other algorithms with similar objectives in Chapter 6, the author has been able to confirm to what extent the original hypothesis is correct and which shortcomings exist.

1.2 Biological Context

Although the theory and research methodology of this dissertation is embedded in CS, the research was inspired by molecular biologists who were discovered to be visually scanning genetic sequences in search of microsatellites. Therefore, this research also contributes to Bioinformatics, a subset of the subject discipline Computational Biology. Computational Biology is defined by BISTIC[8] as the development and application of data-analytical and theoretical methods, mathematical modeling and computational simulation techniques to the study of biological, social and behavioural systems. Bioinformatics is defined by Luscombe *et al.* (2001) as the application of computational techniques to understand and organize the information associated with biological macromolecules. Bergeron (2003) mentions that Bioinformatics distinguishes itself from other scientific endeavors in the sense that it focuses on the information encoded in the genes and how this information affects the universe of biological processes. In order to understand the biological context of microsatellites and other competitive software packages, reflective research[9] has been conducted. However, the main focus of

[4]Computer Theory constitutes mathematical models that describe with various degrees of accuracy parts of computers, types of computers and similar machines [Cohen (1997)].

[5]FAs are defined in detail in Section 4.2.

[6]The term microsatellites is defined in detail in Section 2.2.

[7]The term effectively is discussed in Section 2.6.

[8]The Biomedical Information Science and Technology Initiative Consortium of the National Institution of Health of the United States Of America [National Institute of Health of the United States of America. BISTIC Definition Committee (2000)].

[9]Reflective research includes the systematic and persistent inquiry into already existing knowledge [Kressel (1997)].

this dissertation is the development of a data-analytical and theoretical algorithm to contribute to analyzing of deoxyribonucleic acid (DNA).

DNA is the polymer molecule[10] that stores genetic information of organisms [Paces (2001)]. An organism's genetic information is encoded in DNA as a sequence of four different nitrogenous bases on a sugar-phosphate backbone. DNA can adopt various conformations, including the double helix structure. The four nitrogenous bases (nucleotides) are Adenine (A), Cytosine (C), Guanine (G) and Thymine (T). The sequence of the four nitrogenous bases mirror each other, in each strand of the double helix mirror, in a predefined manner: Adenine on the one strand always binds with Thymine on the other, and Cytosine always binds with Guanine [Bergeron (2003)]. Therefore, the sequence ATTGCA will occur as TGCAAT on the complementary strand of the helix [Paces (2001)]. Examples of genetic databases that store nucleotide sequences include GenBank[11], DDBJ[12], EMBL[13], MGDB[14], GSX[15], NDB[16] [Bergeron (2003)] and GeneCards [Safran *et al.* (2002)]. These databases store the described single-letter symbols — A, C, G and T concatenated.

DNA molecules are subject to numerous mutational events. One of the consequences of these events that can be detected by computationally analyzing genome sequences, is tandem duplication. In tandem duplication a stretch of DNA, which we call a *motif*, is converted to one or more "copies", each following the preceding one in a contiguous fashion. These copies may or may not be exact.

A perfect tandem repeat (PTR) is a string of nucleotides in a genomic sequence whose initial substring (of some arbitrary length), is followed by one or more exact copies of that substring.

In contrast, an approximate tandem repeat (ATR) is a genomic sequence whose introductory substring (or motif) is followed by one or more substrings, of which at least one need not necessarily be an *exact* copy of the motif. The extent to which these non-exact copies may vary from the motif is limited, as will be discussed later in this dissertation.

Short tandem repeats or so called microsatellites (for a detailed definition see Section 2.2) serve as markers in a variety of different genetic studies in plant and animal species [Kim *et al.* (2000)]. Short tandem repeats occur less abundantly in plant genomes than in mammalian or insect systems. Researchers are focussing on the identification of short tandem repeats that occur on different plant genomes

[10]Polymer molecules are large molecules consisting of repeated chemical units joined together.
[11]One of the largest public sequence databases.
[12]DNA DataBank of Japan.
[13]European Molecular Biology Laboratory.
[14]Mouse Genome Database.
[15]Mouse Gene Expression Database.
[16]Nucleic Acid Database.

for various reasons. One of these reasons is to feed an overpopulated planet. At the current birthrate the world population will grow to an estimated number of 12 billion people as the year 2050 is approaching. The consequence of the above mentioned growth is that less and lower quality agricultural land will be available. A highly sophisticated knowledge of plant Biology may allow the development of agronomically important species suitable for producing more food in spite of less available agricultural land [Theologis (2001)]. Better crops can also be assured by breeding cultivars that are disease resistant. Kim *et al.* (2000) proposes in this regard that rice blast disease is one of the most damaging crop diseases world wide. Effective breeding for blast resistant rice cultivars is not easy to achieve as a consequence of the frequent turnover of its pathogenecity [17]. In order to contribute to the breeding of blast resistant rice, molecular biologists have isolated short tandem repeats (microsatellites) to obtain the genomic fingerprint of the pathogenic fungus rice blast disease (Magnoporthe grisea) and their homologous[18] sequences in other fungi, yeast and plant species [Kim *et al.* (2000)].

Given the importance of known and potential biological roles for short tandem repeats (microsatellites) that have been briefly outlined above, it seems essential to develop an efficient and sensitive algorithm to detect these repeats, so that they may receive further study.

Although two of the FireμSat-algorithms[19], that are proposed here, can in theory be applied to search for TRs of any length, the focus at this stage is to introduce algorithms that search specifically for microsatellites. The term microsatellites is defined in Section 2.2. Note, the name allocated to the set of FireμSat-algorithms is FireμSat. There will be referred to FireμSat if the intention is to refer to FireμSat$_1$, FireμSat$_2$ and FireμSat$_3$ at once.

It will be seen that FireμSat has several parameters that can be used to tune its search. It should be emphasized that these parameters have been devised in consultation with the intended user community, who have been unable to usefully deploy existing software for TR detection, and who are consequently scanning sequenced DNA by visual examination. The objective of this dissertation is to develop an algorithm that can be fine-tuned so that redundant data is avoided, and relevant data is not missed during microsatellite detection.

[17] A fungus or bacteria that has the ability to cause a disease.

[18] Protein or nucleotide sequences are homologous if they show a significant level of similarity.

[19] Three different algorithms FireμSat$_1$, FireμSat$_2$ and FireμSat$_3$ are proposed. All these algorithms rely on the implementation of FAs to search for microsatellites.

1.3 Dissertation Layout

The remainder of this dissertation is laid out as follows. In Chapter 2, various background matters are explained and a literature overview of algorithms that aim to detect tandem repeats in DNA is provided. In Chapter 3 two algorithms that deal in an effective manner with the detection of microsatellites (as defined in Section 2.2) in DNA are identified namely, Tandem Repeats Finder [Benson (1999)] and STAR (Search for Tandem Approximate Repeats) [Delgrange & Rivals (2004b)]. From literature overviews at the time this research commenced STAR and TRF appeared to be the most prominent software within the domain of software searching for microsatellites. These packages both detect PTRs as well as ATRs. In 2007 the software package IMEx [Mudunuri & Nagarajaram (2007)] has been released. Although IMExs is also considered to be a prominent software package that detects microsatellites, allowing for mismatches, insertions and deletions, it is not discussed at the same level of detail as TRF and STAR. However, a comparison between the data IMEx generates and the data generated by FireμSat can be found in Chapter 6 of this dissertation. "Effective" in the context of this dissertation, firstly implies algorithms that detect microsatellites containing substitutions and/or deletions and/or insertions. It also implies that the relevant algorithms have a running time smaller or equal to $O(np + nlogn)$, where n is the length of the genetic sequence file under investigation and p is the length of the motif to be detected. Similarly to FireμSat (the proposed algorithm here), STAR detects microsatellites only, whereas Tandem Repeats Finder detects microsatellites as well as minisatellites (defined in Section 2.2) and satellites (defined in Section 2.2).

To illustrate the layout of the dissertation a diagram is provided in Figure 1.1.

The required input and output of Tandem Repeats Finder (Section 3.1) and of STAR (Section 3.2) is not trivial. Therefore it is discussed in detail in Chapter 3. In Chapter 4, the theoretical background as well as the specifications of the algorithm proposed in this dissertation, FireμSat, will be presented. Chapter 5 constitutes a discussion of the input and output of FireμSat. The input of FireμSat is discussed in a similar manner to the discussion of that of Tandem Repeats Finder and STAR in Chapter 3. In Chapter 6 the three algorithms Tandem Repeats Finder (Section 3.1), STAR (Section 3.2) and FireμSat$_2$ (one of the algorithms proposed in this dissertation) are compared. The comparison of the three algorithms is guided by the criteria introduced in Chapter 2, Section 2.6. Software tools should comply to these criteria, if they are to search effectively for microsatellites. Chapter 7 concludes this dissertation and points to future research possibilities.

Note further that a glossary is provided at the end of the dissertation.

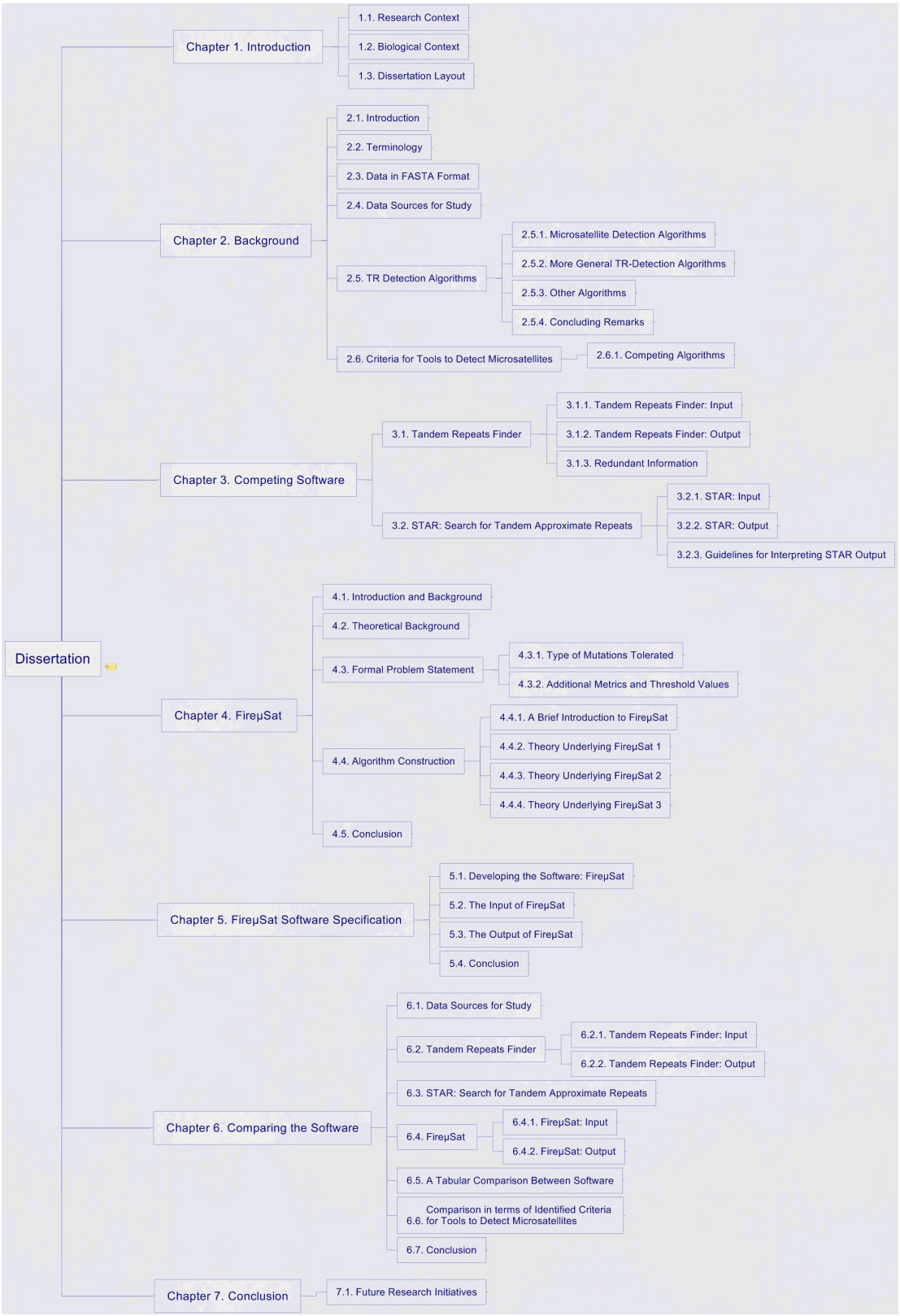
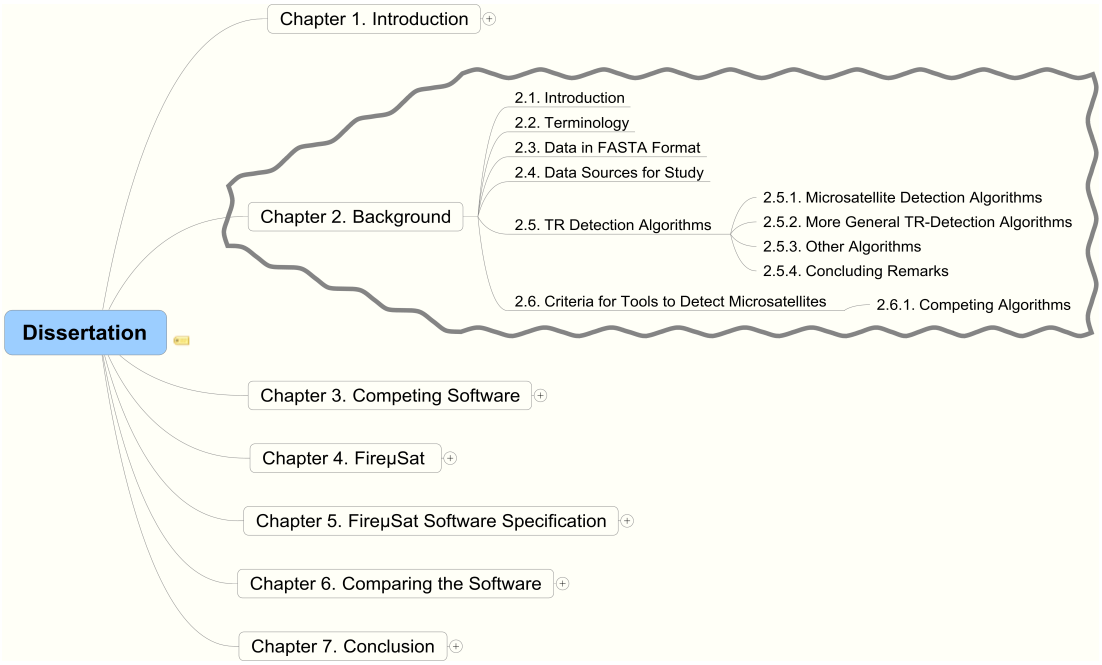


Figure 1.1: An Illustration of the Dissertation Layout

Chapter 2

Background



2.1 Introduction

The purpose of this chapter is to provide a literature overview of different algorithms that directly or indirectly detect tandem repeats on DNA. However, before doing so various background matters are explained.

The first background matter relates to the fact that there is significant inconsistency in regard to the terminology that researchers and software developers use to describe tandem repeats. Therefore, in the interests of consistency, Section 2.2 defines some of the terminology that will henceforth be used. Note that the section does not exhaustively cover all the terminology needed in this dissertation. It is restricted to general terms. Other terminology will be defined when appropriate.

Secondly, it will be seen that most of the algorithms to be investigated in Section 2.5 require that the data to be processed should be in the so-called FASTA format. Section 2.3 explains this format.

It is followed by a discussion of the data that has been used to evaluate the performance of the identified competing algorithms and FireμSat (Section 2.4).

Thereafter Section 2.5 outlines various algorithms that detect repeats, either directly or indirectly. These algorithms were identified after a search of the literature and Web sources. In this regard Van den Bergh (2006) mentions that although most authors reference a selection of software, developed before the software that they propose, there does not seem to be a comprehensive catalogue of relevant software. The algorithms described include the most prominent in the problem domain. An overview of these algorithms will therefore provide insight into the various computational and algorithmic techniques used to address the computational challenge of searching genetic sequences for TRs.

Finally in Section 2.6 criteria with which tools should comply, if they are to search effectively for microsatellites (see Section 2.2 for a definition of microsatellites) are introduced.

2.2 Terminology

In the literature a distinction is made between interspersed repeats and tandem repeats [Pestronk (2005)]. Interspersed repeats are repeated DNA sequences located at dispersed regions in a genome. These repeats are also known as mobile elements or transposable elements. An interspersed repeat occurs if a stretch of DNA (sequence of nucleotides) is copied to a different location through DNA recombination [Lee (1996)].

In contrast with interspersed repeats, a tandem repeat (TR) in a genomic sequence is a string of nucleotides that is characterized by a certain motif which introduces the string, followed by at least one "copy" of the motif. Consider the motif `ACG` with motif length 3 ($|motif| = 3$). Then `ACGACGACGACGACG` is a TR. If the copies of the motif are exact, this text will refer to the TR as a *perfect* tandem repeat (PTR), as in the case of the above example; otherwise (in the case of the

inclusion of non-exact copies in the TR) this text will refer to an *approximate* tandem repeat (ATR). An ATR is thus a string of nucleotides repeated consecutively at least twice with small differences between the instances. An example of an approximate tandem repeat is: `ACGACTACGACGAC`.

In the absence of further qualification, a reference to a TR should be construed as a reference to either a PTR or an ATR. A TR element (TRE) that matches the identified motif of the TR will be referred to as a PTR element (PTRE). A TRE that does not match the motif is referred to as an ATR element (ATRE).

In the literature [Thurston & Field (2005); Myers & Sagot (1998); Delgrange & Rivals (2004b); Lee (1996)] a distinction is made between TRs that constitute microsatellites, minisatellites and satellites. However, terminology is not used consistently in the literature. Castelo *et al.* (2002) coins the term *Simple Sequence Repeats (SSRs)* for microsatellites; Tran *et al.* (2004) terms microsatellites *short tandem repeats*. Delgrange & Rivals (2004b), Benson (1999) and Abajian (2003) consider TRs characterized by motif lengths greater than or equal to two and smaller than or equal to five ($2 \leq |motif| \leq 5$) to be microsatellites. Thurston & Field (2005) and Lee (1996) consider TRs microsatellites if $2 \leq |motif| \leq 6$.

Benson (1999) refers to all other TRs as *variable number tandem repeats* (VNTRs) whereas Delgrange & Rivals (2004b) and Myers & Sagot (1998) also distinguish between minisatellites and satellites. Delgrange & Rivals (2004b) define minsatellites in terms of motif length such that ($7 \leq |motif| \leq 100$) and satellites such that $|motif| \geq 101$.

In this dissertation a microsatellite is considered to be a TR with a PTRE or motif such that $2 \leq |motif| \leq 5$. Microsatellites will include PTREs and ATREs. Thus microsatellites should only be construed as PTRs if this is specifically stated.

However, TRs can only be detected after sequencing[1] of a genome or fragment of DNA has taken place. Sequences that are investigated in order to detect TRs may either be stored in genetic data banks (Section 2.3) or may be generated by various methods, such as the BAC-to-BAC method, shotgun method, Random Amplified Microsatellites (RAMS) method or the Fast Isolation by AFLP[2] of Sequences Containing Repeats (FIASCO) method [Zane *et al.* (2002), Trivedi (2000) and Van der Nest *et al.* (2000)]. Sequencing involves complicated laboratory techniques — an explanation of which is beyond the scope of this dissertation.

After sequencing, the generated data that is to be processed by software packages should be converted to an internationally acceptable format. Most of the software

[1] Sequencing is the determination of the order of nucleotides in a DNA or RNA molecule or the order of amino acids in a protein [Lefers (2004)].

[2] AFLP is an abbreviation for amplified fragment length polymorphism (the occurrence of different forms of a gene in members of the same specie) [Ware *et al.* (2005)].

that detects TRs on genomic sequences has been designed to process data in so-called FASTA format. The two algorithms that we have investigated as competing algorithms in Chapter 3 also require their genetic input sequences to be in FASTA format.

So does the software implementation of the FireμSat algorithm proposed in Chapter 4 of this dissertation.

Therefore, the next Section, Section 2.3, elaborates on FASTA format.

2.3 Data in FASTA Format

The FASTA format defines the file format used to store and exchange information between genetic databases [Farlex Inc. (2005)]. FASTA is used by, amongst others, the European Molecular Biology Open Software Suit (EMBOSS) [Rice & Bleasby (2009)], the Norwegian Bioinformatics platform [Biology Online (2005)] and the National Center of Biotechnology Information (NCBI) that is situated in the United States of America.

The NCBI has developed the widely used Entrez Search System, which allows for the retrieval of a wide range of molecular biology data and bibliographic citations[3].

The genetic database files that are accessed and delivered by this system include nucleotide databases that are in FASTA format.

A sequence in FASTA format begins with a single line description, that is followed by lines of sequence data. The description line is distinguished from the sequence data by means of a greater-than symbol,“>”, in the first column of the first row. It gives a name and/or a unique identifier to the sequence. The description line may also give other relevant information.

For example, it may give an indication of the fragment of DNA that is represented by the data.

FASTA format files often have extensions like .fa, .mpfa or .fsa. Other extensions are, however, also allowed [Farlex Inc. (2005)]. Multiple genetic sequences, all in the same file, each introduced by single description lines are also acceptable.

In general, sequence analysis programs require protein sequences and DNA sequences to be represented in the standard International Union of Biochemistry

[3]Some of the databases that can be searched and accessed via the system include: integrated nucleotide databases (e.g. GenBank, DDBJ and EMBL); protein databases (e.g. SwissProt, PIR, PRF and PDB); and various bibliographic databases (e.g. PubMed databases) [Google (2005)]. A list of these abbreviations can be found in the Appendix.

(IUB) or the International Union of Pure and Applied Chemistry (IUPAC) amino acid and nucleic acid codes.

FASTA format conforms to these codes. These codes prescribe the use of letters over the alphabet $\Sigma = \{A, C, G, T\} \cup \{N, X\}$. The IUB and the IUPAC allow two exceptions pertaining to DNA: lower case letters are accepted, but are mapped to the corresponding upper-case letters. Additionally, a single hyphen can be used to represent a gap of length one[4].

In general, a gap can be of indeterminate length. Before submitting a request to sequence analysis programs, any numerical digits[5] in a query sequence should either be removed or replaced by appropriate letter codes (e.g., N for unknown nucleic acid residue or X for unknown amino acid residue) [The Marine Biological Laboratory (2003)]. An example of a genetic sequence in FASTA format is given below.

```
>Cow
ATGGCATATCCCATACAACTAGGATTCCAAGATGCAACATCAC
CTTAAGCTTCGACTCCTACATAATTCCAACATCAGAATTAAAG
CCCGTCCAGGCTTATATTACGGTCAATGCTCAGAAATTTGCGG
GTCAAACCACAGTTTCATACCCATTGTCCTTGAGTTAGTCCCA
CTAAAGTACTTTGAAAAATGATCTGCGTCAATATTA-------
--------------TAA
```

This example was obtained from the web site of *The Marine Biological Laboratory* [The Marine Biological Laboratory. Workshop on Molecular Evolution (2004)]. For the purposes of evaluating and comparing one of the proposed algorithms (FireμSat$_2$) with competing software, two sets of data in FASTA format are used. Next the details of these sets of data are discussed.

2.4 Data Sources for Study

In later chapters the FireμSat algorithms for microsatellite detection will be explained and compared against two of the most prominent competing algorithms. Two sets of data were used to compare these algorithms in terms of their practical execution. They are the following:

[4] A gap of length one could occur within a DNA sequence if, for example, for a certain position it is unclear which nucleotide is prominent during the final step of sequencing [National Institute of Health of the United States of America. BISTIC Definition Committee (2000)].

[5] Numerical digits are generated during one of the final steps of sequencing to indicate the prominence of a certain nucleotide in a specific DNA sequence position.

1. *Data of the fungus* ***Fusarium Graminearum***

   Data of this fungus was made available by the Center of Genome Research. It constitutes 33 930 392 bytes. The data is in FASTA format and thus consists of concatenated characters over the alphabet $\Sigma = \{A, C, G, T\}$. The base pairs of the *Fusarium Graminearum* genome has been divided in different so called scaffolds or attached regions. The number of base pairs varies for each scaffold. The second scaffold is labeled with the heading > *Fusarium graminearum 1.54 (scaffold 1).*

2. *Data of the* ***Cylindrocladium Pauciramosum***
   The genomic data of the *Cylindrocladium Pauciramosum* was generated by Dr. L.P. Wright, one of the molecular biologists who was visually scanning DNA sequences with his eyes to detect microsatellites. Wright reports on his findings based on the microsatellites he detected in Wright *et al.* (2007). The Random Amplified Microsatellites (RAMS) method was applied in order to obtain the generated data[6]. The format of the data obtained is relevant to the present discussion.

   In contrast to the fusarium genome that consists of an approximately 33 mega bytes of concatenated characters of the alphabet $\Sigma = \{A, C, G, T\}$ divided by scaffolds, the data generated by the RAMS method consists of various short sequences, each containing data of different so called "librarie" (explained in the itemized list below) and having different sequence headers. The following gives examples of two of the genetic sequences generated by Wright *et al.* (2007), these sequences are contained in the same file:

```
>BL141
CCACCACCACCACCAACACAATTGCACCGCTAGTGGCTATATTTGATGCC
CTCAAAATTCCCGCACCGTGGGCACCAGAGGCCAAGGATTTCGACTACGC
AAACACGACTTTGCTGATTATGGGTGGTGGATCCAGCACCGGCAAATTTG
GCGTACAATTAGCCAAGTTAGCAGGCATTGGCAAGATTGTTGTTGTTGTT
G
>BL204
ACAACAACAACAACAGTAGGAAGGAAATAATAACCATAGAAACCAGTTTT
TGAAATCTGTTCAGTTTAATTAATTTTGCAATTTTTATTCCCTATTTTTG
TCTAGAGCGTAGGAGTGTAAGTGGATCTCGCATCCTTCGAGAGCCCGCTG
CTCATTACGGAGACCCTCCCTCCACTCTTTGCTTGCTTCAACAAGGTTGA
ACTTTGTTGCGGCTGTCTCTCTCTAGTCTTTTGATGCCATCGCTCATTTT
```

[6]The RAMS method is a fairly complicated technique and the details thereof is, as mentioned in Section 2.2, beyond the scope of this dissertation. A description of the technique can be found in [Van der Nest *et al.* (2000)].

```
GGCCTTTCCCGCGGTCTGTTGCCGCCTTCTCCTTATCTCCTGTGTGTGTG
TGTGTG
```

The reason for the different sequences can briefly be explained as follows:

- During the enrichment of the genetic substance that contains fragments of the organism of which genetic data is generated, the molecular biologist obtains a whole mixture of different DNA fragments. These different DNA fragments are referred to as a *library*.
- The RAMS method uses a cloning step in one of the final steps in order to separate different DNA fragments. Cloning entails the incorporation of the respective DNA fragments into respective plasmids[7] by means of a process called ligation[8].
- Transformation is the next step and occurs when the respective plasmids are introduced into bacteria cells.
- With the growth of the bacteria cells the plasmids are also reproduced. After 24 hours every single bacteria cell will have multiplied into billions of cells. Each of these cells will contain the various plasmids with their respective DNA fragments.
- The plasmids are then isolated from the cells to be used in further DNA manipulation work, *inter alia* for sequencing. Thus sequences created by means of the RAMS method entails the generation of sequences of different fragments of DNA.

It is in the interest of the molecular biologist that different relevant fragments of genetic data in the original *library* should be distinguishable. The sequenced fragments are relatively short DNA sequences in FASTA format. It is not practical (and it is very time consuming) to create separate files for each fragment of DNA. A more practical manner of storing the genetic data of each of the original fragments is by compiling a large number of DNA fragment sequences into one file. One should then distinguish between the respective fragment sequences obtained by means of a single line description of the fragment followed by the fragment itself, as is allowed in FASTA format.

Consequently, in terms of useability, software should make provision for the processing of multiple genetic sequences, with their respective descriptions, in one file.

[7] A plasmid is any replicating DNA element that can exist within the cell independently of the chromosomes. Synthetic plasmids are used for DNA cloning [Bio-Synthesis, Inc (2007)].

[8] Ligation refers to the process of splicing two pieces of DNA together — details of the process is not relevant to the current discussion.

Now that the file format and the data to be investigated have been explained, various software applications that detect TRs directly or indirectly will be explored in Section 2.5.

2.5 TR Detection Algorithms

There are a variety of software applications that meaningfully support intra- and inter-genetic sequence analysis. Several of these software applications that support Molecular Biology are freely available online. Organizations that provide links or direct access to this software include:

- EMBOSS: The European Molecular Open Source Software Suite [Rice & Bleasby (2009)].
- NCBI: National Center for Biotechnology Information [National Center for Biotechnology Information (2005)].
- The BROAD institute [Broad Institute (2009)].
- BCCL: Bioinformatics Computational Core Laboratories [Bioinformatics Computational Core Laboratories (2005)].
- TEXTCO [Hackett & Gross (2007)].

These web sites include various software packages that search for repeats on genomic sequences.

As mentioned in Section 2.1, academics who publish their developments of TR-detecting software usually reference other, already published, software. Some of these criticisms will be mentioned below after the discussion of the selected software. In this regard one should take note thereof that none of the existing publications have been found to be exhaustive in the sense that the authors were describing all the available tools [Van den Bergh (2006)].

Similarly, it is not the intention of this study to present an exhaustive literature overview of all the available tools. To the best of the author's knowledge, the algorithms described include the most prominent in the problem domain. The algorithms also contribute to the provision of an overview constituting the different needs within the knowledge framework of the detection of TRs on genetic strings. In this study computational, algorithmic techniques by which academics address the computational challenge of searching genetics sequences for TRs are also addressed.

In general, this study did not involve a re-investigation of all the software already referred to and criticized by other authors. Instead it was decided to focus on algorithms that address some of the needs of the molecular biologists, described in terms of various criteria in Section 2.6. However, if this investigation were to be limited to existing software that fully addresses the particular needs of the molecular biologist and that complies with the criteria stipulated in Section 2.6, then it is doubtful whether anything would be found.

A number of software packages that detect tandem repeats, either directly or indirectly, are investigated even though they are not fully compliant with the features mentioned in Section 2.6.

In this regard STAR [Delgrange & Rivals (2004b)] was included, which searches for microsatellites in particular, allowing for substitutions, deletions as well as mismatches, although it has been criticized by Wexler *et al.* (2005). STAR also provides its user with statistical relevant information.

Tandem Repeats Finder [Benson (1999)] has been compared with both ATR-Hunter [Wexler *et al.* (2005)] and STAR [Delgrange & Rivals (2004b)], with the authors of the latter two systems each claiming that their respective algorithm is better than Tandem Repeats Finder in terms of accuracy. Nevertheless, Van den Bergh (2006) mentions that Tandem Repeats Finder remains the most cited TR-detecting software. For this reason, Tandem Repeats Finder has also been included as part of this literature overview. It is also worth noting that Tandem Repeats Finder can be used much more easily to execute computations on larger files than either ATRHunter and STAR. (Details of this statement in relation to ATRHunter can be found in Van den Bergh (2006); details regarding STAR are to be found in Chapter 6 of this dissertation.)

For the purposes of this dissertation, a detailed discussion of each of the algorithms classified below will not be given. Instead they will be discussed in terms of:

- their computational technique used to detect microsatellites; and
- some of the shortcomings of the respective packages as identified from the perspective of a molecular biologist.

A number of these software packages (including STAR, Tandem Repeats Finder and ATRHunter, all of which are discussed below) implement alignment algorithms in order to detect TRs. Thus before the different algorithms are discussed the concept “alignment” as a string comparative technique will briefly be explained in terms of an example.

Within the context of Molecular Biology, an alignment refers to the comparison of two genetic strings. Camp *et al.* (1998) describes alignment as the loose positioning of one string relative to another string. An example of an alignment where the top row represents the first genetic sequence and the second row represents the second genetic sequence is:

```
T T C T T C T T C C T C T T
T T C T T C T T C T T C T T
```

Alignments can be evaluated quantitatively by means of an alignment score. For example, suppose a scoring function is defined as follows:

$$Score = (number\_of\_matches \times match\_score) + (number\_of\_mismatches \times mismatch\_penalty) + (number\_of\_indels \times indel\_penalty)$$

Then, letting $match\_score = 2$ for each match; $mismatch\_penalty = -2$ for each mismatch and the $indel\_penalty = -3$ for each indel (insertion or deletion) then the score of the above alignment can be calculated as follows:

$$\begin{aligned} Score &= (13 \times 2) + (1 \times (-2)) + (0 \times (-3)) \\ &= 26 - 2 + 0 \\ &= 24 \end{aligned}$$

The determination of an alignment score of two strings is thus relatively simple. However, it is very challenging to determine the optimal alignment of two sequences. The optimal alignment of two sequences implies the determination of an alignment that will obtain the highest score from aligning two given sequences in all possible manners [Van den Bergh (2006)].

Various software packages will be discussed below, some of which implement alignment algorithms.

They are divisible into the following three categories.

1. Algorithms searching specifically for microsatellites:
    - Sputnik (Section 2.5.1.1)
    - TROLL: Tandem Repeat Occurrence Locator (Section 2.5.1.2)

- Msatminer (Section 2.5.1.3)
- STAR: Search for Tandem Approximate Repeats (Section 2.5.1.4)
- IMEx As mentioned IMEx has only been made available after most of this study has been completed. A comparison between the data generated by IMEx and FireμSat can be found in Chapter 6, Section 6.5.1.

2. Algorithms that detect microsatellites as a subset of tandem repeats of various length:
   - ATRHunter (Section 2.5.2.1)
   - FORRepeats: detects repeats on entire chromosomes and between genomes (Section 2.5.2.2)
   - Tandem Repeats Finder (Section 2.5.2.3)
3. Algorithms that do not specifically aim to detect tandem repeats, but that detect tandem repeats indirectly in some manner:
   - REPuter: fast computation of maximal repeats in complete genomes (Section 2.5.3.1)
   - RepeatFinder (Section 2.5.3.2)
   - BLAST: Basic Local Alignment Search Tool (Section 2.5.3.3)
   - RepeatMasker (Section 2.5.3.4)

2.5.1 Microsatellite Detection Algorithms

In this section software packages detecting microsatellites *per se* are discussed.

2.5.1.1 Sputnik

Sputnik uses a recursive algorithm to search for repeated motifs of nucleotides of length between 2 and 5. Sputnik reads through the entire submitted genetic sequence, assumes the existence of a repeat at every position, compares subsequent nucleotides and applies a simple scoring rule. Mismatches and indels are tolerated but affect the overall score. If the resulting score rises above a preset threshold then the TR along with its threshold becomes part of the output. If the threshold score falls below a cutoff threshold then the search is abandoned and restarted at a next nucleotide. Each nucleotide that matches the value predicted by assuming a repeat increases the score. Each *error* decreases the score. If an

error is encountered then at least one of the three possible kinds of errors *mismatch, insertion* and *deletion* is assumed and recursive calls to the comparison routine are made. If the resulting score from one of these errors is above the cutoff threshold, then it is returned and the best of three is pursued [Abajian (2003)].

Besides the fact that Sputnik does not contribute to the statistical analysis of detected TRs it does not detect all microsatellites. In contrast with the STAR algorithm (Section 2.5.1.4), the developer cannot claim that Sputnik is an exact algorithm. We have run Sputnik and found, for example, that both Tandem Repeats Finder (Section 2.5.2.3) and STAR (Section 2.5.1.4) detected two TRs[9] which Sputnik failed to detect. Consequently we decided not to further evaluate Sputnik as a competing algorithm to FireμSat.

2.5.1.2 TROLL: Tandem Repeat Occurrence Locator

TROLL is a microsatellite finder based on the *Aho Corasick* Algorithm (ACA) that was developed to solve the so-called *dictionary problem* i.e. to find all occurrences from a list of patterns in a text string. The ACA uses a *keyword tree* to find all occurrences of any pattern from a set in a text string. The *keyword tree* also stores information regarding pattern similarity by means of *failure links*. If the algorithm encounters a partial match in the text, it uses the *failure links* to continue the search without re-sampling characters in the text, thereby achieving linear time complexity.

TROLL uses the ACA to detect pre-selected patterns in a text string and keeps simultaneously track of tandem repetitions so that microsatellites can be detected. In the context of Bioinformatics, patterns correspond to motifs and the text string is a DNA sequence [Castelo *et al.* (2002)]. Castelo *et al.* (2002) mentions, that the sole purpose of TROLL is to find perfect tandem repeats. This is in contrast to the algorithm presented, which also detects approximate tandem repeats. Therefore TROLL will not be evaluated as a competing algorithm to FireμSat.

2.5.1.3 Msatminer

Msatminer is a package of perl scripts[10], released under *GPL*[11]. Msatminer comprises of four core scripts plus additional supporting scripts. The four core scripts

[9] with $|motif| = 3$ on scaffold 41 of the *Fusarium Graminearum* genome (Section 2.4).

[10] Perl is the abbreviation for *Practical Extraction and Report Language*. Perl is a programming language especially designed for text processing [Webopedia (2004)].

[11] GPL is an abbreviation for the General Public License of GNU (GNU is a recursive acronym for *GNU's Not UNIX*) [Johns (2005)].

are:

- **Msatfinder**
  Msatfinder is the main script in the Msatminer package. Msatfinder is responsible for the detection of microsatellites and the generation of the output file that can be used by other scripts or applications. Msatfinder implements regular expression[12] matching to locate perfect tandem repeats.

- **Msataligner**
  Msataligner takes FASTA files (Section 2.3) that were generated by Msatfinder containing microsatellites and their flanking regions and performs two functions:

  – Comparing all microsatellites against all relevant genomes (whether a genome is considered relevant depends on the molecular biologist) with bl2seq[13].

  – Blasting[14] all detected microsatellites against all others with NCBI blastall[15] and generating alignments or trees that can be viewed using an editor such as Jalview[16].

- **Msatannotator**
  Msatannotator is used to compare microsatellite polymorphism[17] between two closely related genomes and to generate bl2.seq. between the two submitted sequences [Gish (2009)]. of the two genomes. The user is also allowed to annotate polymorphisms found. Additionally the script generates a table summarizing the microsatellites and their degree of conversion[18].

[12]Regular expressions are defined and explained in Chapter 4, Section 4.2.

[13]Bl2seq performs a comparison between two sequences using either the blastn (blast nucleotides (Section 2.5.3.3)) or the blastp (blast peptides (Section 2.5.3.3)) algorithm in order to generate a summary of the regions that are conserved (in the context of Molecular Biology conserved microsatellites on genes refer to microsatellites that are to be found in corresponding positions in different genomes [Chao *et al.* (1993)])between several genomes.

[14]Blasting, within the context of the BLAST-software package (Section 2.5.3.3) refers to the comparison of genetic sequences. Typically, if a "query" genetic sequence is submitted then BLAST searches genetic data banks (Section 2.3) for all sequences that are very similar to the so-called, usually newly sequenced, "query" sequence [Gish (2009)].

[15]NCBI (National Center for Biotechnology Information) blastall should directly be obtained from NCBI and is not part of the Washington University BLAST software package.

[16]Jalview is a multiple alignment editor, like Seaview, Cinema and Belvu but written entirely in Java. It is widely used in a variety of web pages (e.g. the EBI Clustalw server and the Pfam protein domain database). It is also available as a general purpose alignment editor [Clamp (2009)].

[17]Polymorphism indicates the occurrence of different forms of a gene in members of the same species [Ware *et al.* (2005)].

[18]Gene conversion is a process that is often associated with recombination a process during which one allele is replicated at the expense of another.

- **Msatviewer**
  Msatviewer is used to provide online access to the databases that are created as the output of Msatfinder [Thurston & Field (2005)].

Msatminer firstly detects perfect microsatellites by means of Msatfinder. Thereafter Msatminer allows the user the possibility of performing various comparisons on the data obtained. Msatminer detects PTRs whereas our proposed algorithm detects ATRs and PTRs. Furthermore, the focus of our algorithm is not on carrying out inter-genetic comparisons but on effectively detecting microsatellites. Therefore, Msatminer was not further evaluated as a competitor for FireμSat. However, Msatminer reflects the need of the molecular biologist to detect correspondence in terms of microsatellites between different genomic sequences, thus inter-genetic comparison in terms of microsatellites. At present the software proposed by this dissertation does not address the issue of inter-genetic comparison, but the possibility of extending the proposed software is recognized as a challenge for future research.

2.5.1.4 STAR: Search for Tandem Approximate Repeats

STAR is an exact algorithm that detects significant microsatellites of a selected motif in a DNA sequence in FASTA format. (In the case of STAR *significance* is assessed by a measure of local compressibility. The concept "compressibility" in the manner STAR defines it, is explained in more detail in Chapter 3 (Section 3.2).)

STAR searches a genetic sequence for motifs that are entered by the user. For a given motif, STAR returns a description of all the microsatellites. It also returns the respective optimal alignments[19] of the various detected microsatellites and the selected motif [Delgrange & Rivals (2004b)].

STAR is one of the algorithms that has been selected to be evaluated as a competing algorithm. STAR will receive further attention in Chapter 3, where it is discussed in more detail. STAR focuses on the detection of microsatellites and provides the user with some biological, statistically relevant data. An elaboration on the selection of STAR is to be found in Section 2.6.

[19]If we assume a function that is implemented to calculate an alignment score then the optimal alignment of two sequences implies the determination of an alignment that will obtain the highest score from aligning two given sequences in all possible manners [Van den Bergh (2006)].

2.5.2 More General TR-Detection Algorithms

The algorithms in this section detect tandem repeats of various length. Microsatellites are therefore detected as a subset of these, their length being in the range $2 \leq |motif| \leq 5$.

2.5.2.1 ATRHunter

The ATRHunter software constitutes two phases: a screening phase, followed by a verification phase. The screening phase generates a list of candidate regions that may contain TRs. Thus the screening phase identifies substrings which each have an unusually high probability of being a TR. The candidate regions are subsequently verified by, amongst others, aligning candidate approximate tandem repeats and by making use of dynamic programming. Dynamic programming as well as the type of dynamic programming that is implemented by the most alignment algorithms (including ATRHunter) is discussed in more detail in Chapter 3 (Section 3.1.1).

Flexibility in terms of approximation is achieved by using a variable size sliding window along with a suitable similarity metric[20]. Details of the mapping of the motif to the ATRE is beyond the scope of the present discussion, but can be found in Wexler *et al.* (2005).

A novel statistical model captures the metric's distribution, describing the probability that a given ATRE corresponds to a given motif [Wexler *et al.* (2005)]. From the results obtained by Wexler *et al.* (2005), ATRHunter seems to be one of the most effective algorithms for detecting TRs at present. Van den Bergh (2006) mentions, however, that she has run ATRHunter but that she could not obtain any output. Van den Bergh (2006) considers that the file length constituting 17.9 megabytes may be problematic as the authors used only a file of 5.5 megabytes while they were comparing ATRHunter with Tandem Repeats Finder. ATRHunter does not provide its user with relevant analyzed, biological statistical data. It only addresses the computational problem of detecting TRs on genetic sequences - ATRHunter will not be evaluated as a competing algorithm.

2.5.2.2 FORRepeats

FORRepeats detects ATRs on entire chromosomes and between genomes. In its first step, it detects exact repeats in large sequences. In its second step, TRs (allowing errors of the detected exact repeats) are computed during a pairwise comparison between two extended exact repeats. The details of the extension of

[20]This metric quantifies the differences between the elements of two sets.

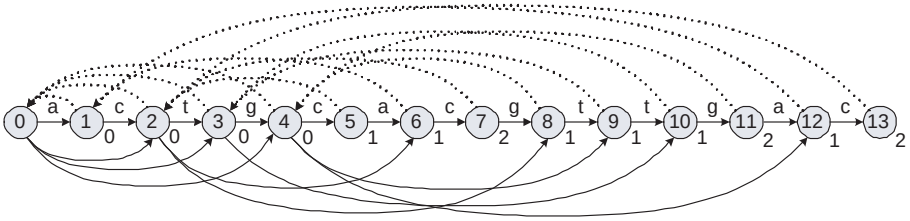


Figure 2.1: The oracle for the word `ACTGCACGTTGAC`

exact repeats the comparison thereof are beyond the scope of this dissertation. However, the interested reader can consult Lefebvre *et al.* (2003) in this regard.

FORRepeats uses a heuristic method that is based on a novel data structure called a factor oracle. Factor oracle technology is an application of finite automata technology. The factor oracle of a word p of length m, denoted by *Oracle(p)*, is a deterministic finite automaton (Q, q_0, F, δ) where $Q = 0, 1, ..., m$ is the set of states, $q_0 = 0$ is the starting state, $F = Q$ is the set of terminal states and δ is the transition function. The factor oracle of a word p of length m has the following properties:

- it has exactly $m + 1$ states,
- its number of transitions is in the range $[m, 2m - 1]$ and,
- it recognizes at least all the factors of p and more words (the exact set of words recognized by the factor oracle is still being investigated).

There exist a bijection between the states of the oracle and the $m + 1$ prefixes of p (including the empty one). Each transition leading to a state i is labeled by $p[i]$. Two kinds of transitions are distinguished namely:

- Internal transitions: transitions from state i to state $i + 1$ and,
- External transitions: transitions from state j to state i where $(j - i) > 1$.

For the above the following must hold: $0 \leq i < j \leq m$. There are exactly m internal transitions. To store the oracle one needs to store only the word p and at most $m - 1$ external transitions without their labels. The factor oracle is a structure that is economical in terms of memory and runtime. The memory used is approximately 10.5 times the length of the sequence — the structure is linear in terms of memory. The construction of the structure is also linear in terms of runtime [Lefebvre *et al.* (2003)]. See, Figure 2.1.

Lefebvre *et al.* (2003) conducted experiments to determine the accuracy of FORRepeats. These experiments indicated that FORRepeats performs better at detecting repeats that are longer than 20 base pairs. The performance of FORRepeats, in terms of the more accurate detection of longer repeats than shorter repeats, is comprehensible if one consider the calculation technique which is implemented by FORRepeats. Firstly exact repeats are found then these repeats are extended to the left and to the right such that the repeats become approximate repeats for which certain threshold values hold. If the repeats are very short then a large number of exact repeats will be found which will be time consuming to extend and may result in overlapping.

This implies that FORRepeats is well-suited for the detection of minisatellites or duplicated genes. FORRepeats does not detect microsatellites that effectively [Lefebvre *et al.* (2003)].

Therefore, in spite of the good performance of FORRepeats, this study will not explore the algorithm further.

2.5.2.3 Tandem Repeats Finder

Tandem Repeats Finder models tandem repeats by percent identity and frequency of insertions and deletions (indels) between adjacent motif copies. The program uses statistically based recognition criteria. Tandem Repeats Finder consists of detection and analysis components. The detection component uses a set of statistically based criteria to determine *candidate* tandem repeats. The analysis component aims to produce an alignment for each candidate. If the analysis component is successful then it gathers a number of statistics regarding the alignment and the nucleotide sequence [Benson (2003b)]. Tandem Repeats Finder detects microsatellites ($|motif| < 6$) as well as repeats with longer pattern lengths [Benson (1999)]. Tandem Repeats Finder performs well in terms of runtime and memory management. Tandem Repeats Finder also provides its user with relevant biological and statistical data.

Tandem Repeats Finder is the second algorithm that has been identified for evaluation as a competing algorithm. This study's choice of Tandem Repeats Finder is justified in Section 2.6 and is elaborated on in Chapter 3.

2.5.3 Other Algorithms

The algorithms in this section are not specifically aimed at detecting tandem repeats. However, they nevertheless detect tandem repeats indirectly, in some or other manner.

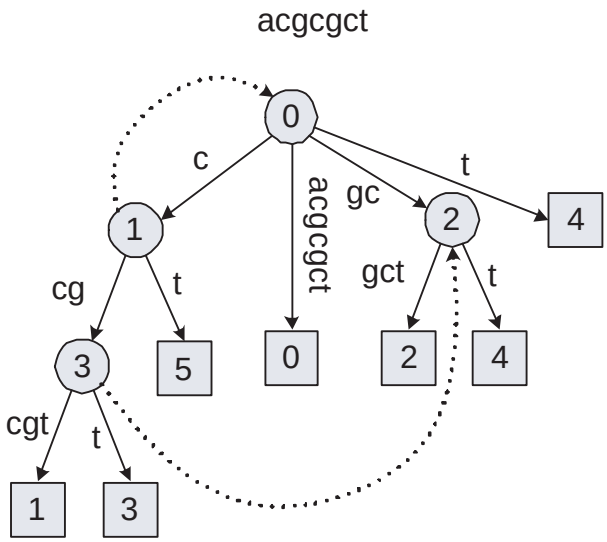


Figure 2.2: A suffix tree for the word `acgcgct`

2.5.3.1 REPuter

REPuter is a software tool that efficiently computes exact repeats and palindromes in a complete genome. The repeats computed are of maximal length. REPuter implements suffix trees in order to detect repeated sequences [Kurtz & Schleiermacher (1999)]. Suffix trees are data structures that are used to rapidly solve certain computational problems pertaining to sequence processing. Such problems include, for example, finding all occurrences of short sequences in DNA, and finding common sequences in two different sequences [Hyde (2000)]. Figure 2.2 gives an example of a suffix tree for the sequence `acgcgct`.

The runtime and space requirement of REPuter can be represented as a linear function of the length of the genetic input sequence and of the size of the output [Volfovsky *et al.* (2001)]. REPuter consists of two programs, namely: a search engine; and a visualizing component. The search engine processes a DNA sequence in FASTA-format (Section 2.3) and returns a representation of all maximal perfect repeats in a simple ASCII format. The visualizing component processes the output of the search engine and generates an overview of the number, the length and the location of the repeated substrings. REPuter is a highly efficient computational tool that can find all exact repeats in sequences that are as long as complete eukaryotic chromosomes [21]. Eukaryotic chromosomes may occupy

[21] The term eukaryotic cell refers to a cell that has a nucleus. Chromosomes are the self-replicating genetic structures of cells containing the cellular DNA that bears in its nucleotide sequence the linear array of genes. Eukaryotic chromosomes play an important role in mitosis (cell division) and consist of various DNA sequence elements, including tandem repeats. [Herr

between ten and hundred mega bytes of memory space [Volfovsky *et al.* (2001)].

REPuter has been criticised by Volfovsky *et al.* (2001) for not providing a sufficient overview of the repetitive structure within the genetic sequence where repeats are detected. A repetitive structure will include the organization of detected repeats into classes so that repeat databases can be created [Volfovsky *et al.* (2001)]. However, Repeatfinder, the software to be discussed in more detail below, utilizes the output of REPuter to provide such an overview of detected repeats by assigning appropriate repeats to so-called similarity classes (discussed simultaneously together with RepeatFinder). Since REPuter is limited to the detection of perfect tandem repeats, it will not be treated as a competitor to the FireμSat algorithm.

2.5.3.2 RepeatFinder

RepeatFinder is a computational system that analyzes the repetitive structure of genomic sequences (complete genomes and partial genomes). Its authors propose a clustering method for analyzing repeat data that has been captured in suffix trees. RepeatFinder implements an algorithm that uses suffix trees to detect all the exact repeats in a given genetic sequence. REPuter (Section 2.5.3.1) could be the generator of such suffix trees. Detected repeats are then organized into classes and repeat databases are created. The algorithm implemented by RepeatFinder can be described at a high level as follows:

- Selection and pre-processing
  The list of all the exact repeats (provided as output by, for example, REPuter) is interpreted by RepeatFinder as a partition of the original genome sequence. Each point of the partition has a reference to the applicable pair of coordinates (A_1, A_2) as well as to the length l.

- Merging procedure
  This procedure merges two exact repeats that either overlap or that occur within a limited distance (gap) of one another. The new merged repeats will always have the property that significant subsequences of the repeat appear at least twice in the genome sequence.

- Classification
  The various repeat classes are defined during this step. Each resultant merged repeat (of the previous step) will be assigned to a specific class. The following conventions are used in the classification:

(2008) and Biology Online (2005)]

  - If a repeat contains at least one repeat that already belongs to an existing class then the merged repeat will be assigned to that class.
  - If a merged repeat contains references that belong to multiple distinct classes then those classes are combined into one class.
  - If a merging repeat contains no references to an existing class then the merged repeat forms its own class.

- BLAST searches and repeat class updates
  The initial class classification is based on exact repeats. WU-BLAST (see Section 2.5.3.3) is run in order to compare the different repeat classes, created by the previous steps, with one another. The resultant matches between the various already existing classes, detected by WU-BLAST, serve as input to a procedure of RepeatFinder that updates the appropriate repeat classes, creates new classes and/or eliminate old classes. If the already constructed repeat set contains approximate repeats, then this step may be omitted [Volfovsky *et al.* (2001)].

RepeatFinder does not focus on the detection of microsatellites *per se*. However, it reflects the need to detect and classify TRs. RepeatFinder does not propose a new algorithm, but makes use of existing algorithms and cannot be considered an exact algorithm. RepeatFinder makes use of BLAST searches to update repeat classes. BLAST is developed to report on subject sequences of query that have a large number of exact matches. Typically, if a "query" genetic sequence (a "query" sequence may be any genetic sequence, but is usually a newly sequenced genetic sequence of which information is required in terms of comparisons with other genetic sequences to be found in genetic databanks) is submitted then BLAST searches relevant genetic data banks (Section 2.3) for all sequences that are very similar to the so-called, usually newly sequenced, "query sequence". In the case of RepeatFinder the different created repeat classes are compared to each other in order to determine matches between them that can result in the merger of some of the existing classes. TRs detected by RepeatFinder will therefore also be restricted to those that have a large number of exact matches. Consequently RepeatFinder will not be evaluated as a competing algorithm to FireμSat.

2.5.3.3 BLAST: Basic Local Alignment Search Tool

BLAST does not detect TRs directly, but several other software applications implement BLAST for this task. Some of the relevant software was discussed in Section 2.5.1.3 and Section 2.5.3.2.

For this reason, the prominence of BLAST in Computational Biology (see below), and the fact that finite automata form the theoretical underpinnings of BLAST, it was decided to include some details pertaining to this classic algorithmic implementation.

BLAST was introduced to the academic community in 1990 [Altschul *et al.* (1990); Camp *et al.* (1998)]. The name of BLAST implies the objective of the software tool, namely to find substrings of query and subject sequences that have a large number of exact matches. BLAST rapidly identifies significant matches between, amongst others:

- newly sequenced genetic material and existing databases of nucleotides and;
- newly sequenced proteins and databases of existing amino acid sequences [Camp *et al.* (1998)].

Although BLAST performs well in determining almost exact matches, it is less effective if the approximate matches to be determined are less exact. Herrmannsfeldt (1998) and Camp *et al.* (1998) mention that BLAST is one of the most popular similarity search tools available in the field of Computational Biology. The application BLAST enables researchers to draw meaningful conclusions about the structure and function of sequenced genetic and protein material. The output of BLAST also serves as an indication of worthwhile sequences to be searched by more sensitive, computationally expensive software [Camp *et al.* (1998)].

The theoretical underpinnings of BLAST are realized in the implementation of three steps:

- The compilation of a list of high scoring words.
  The list of words is constructed from the query sequence and depends on a calculated score that should be above a certain threshold value T. The score depends on a matrix of similarity scores, where identities and conservative replacements have positive scores while unlikely replacements have negative scores. (The PAM-120 matrix[22] is used for amino acid sequence comparisons; for DNA sequence comparisons, the value of +5 is assigned to identities and the value -4 is assigned to mismatches. It is possible to change these scores.)
- The scanning of a database for hits.
  The genetic sequences of the relevant database(s) are scanned for the compiled list of high scoring words. Thus the problem *bounces down* to the detection of certain short sequences which may be contained within a long

[22]PAM-120 is a predefined scoring matrix for protein sequences. PAM is an abbreviation for Point Accepted Mutations [Cowen (2002)].

sequence. For the scanning phase, a deterministic finite automaton (DFA) is used. Acceptance is signalled on transitions (Mealy paradigm - Mealy machines are discussed in Section 4.2), as opposed to acceptance on states (Moore paradigm - Moore machines are discussed in Section 4.2), the latter being the approach followed by FireμSat.

- Extending hits.
  In order to find a local maximal segment pair (MSP)[23] a hit detected by the constructed DFA has to be extended. The extension process in one direction is simply terminated whenever a segment pair is reached whose score falls below the best score for shorter extensions.

Since BLAST has been released, computing professionals have been improving its functionality and have introduced new computing features. Washington University BLAST (WU BLAST), version 2, was the latest released at the time of writing this dissertation. The feature list of WU BLAST is still in the process of expanding [Gish (2009)]. The complete suite of search programs collectively referred to as AB-Blast — unified database search programs and their functionality are as follows:

- **blastp:** compares peptide[24] sequence queries to peptide sequence databases;
- **blastn:** compares nucleotide sequence queries to nucleotide databases;
- **blastx:** compares nucleotide sequence queries dynamically translated in all six reading frames. The six reading frames can be explained as follows. Each amino acid constitutes three nucleotides. It is in general impossible to determine the location of the boundaries between the sets of three nucleotides within a genetic sequence. Therefore, three reading frames are translated from a genetic sequence. Consider the genetic substring below:

```
Index:        1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
DNA sequence: T  A  C  C  T  C  T  T  G  C  T  C  A  T  C
```

  If the reading frame starts in position 1 then amino acids constituting the following set are deduced:

[23] A maximal segment pair (MSP) is the highest scoring pair of identical length segments chosen from two sequences.

[24] A peptide is an organic compound composed of amino acids linked together by peptide bonds. The peptide bond always involves a single covalent link between the α-carboxyl (oxygen-bearing carbon) of the one amino acid and the amino nitrogen of the other amino acid [Answers.com (2003)].

$$Frame1 = \{\texttt{TAC}, \texttt{CTC}, \texttt{TTG}, \texttt{CTC}, \texttt{ATC}\}$$

If the reading frame starts in position 2 then the amino acids deduced constitutes the set be low:

$$Frame2 = \{\texttt{ACC}, \texttt{TCT}, \texttt{TGC}, \texttt{TCA}\}$$

If index position 3 is considered as the starting position then the set of amino acids will be as follows:

$$Frame3 = \{\texttt{CCT}, \texttt{CTT}, \texttt{GCT}, \texttt{CAT}\}$$

The reading frame that is used determines which amino acids will be encoded [Cooper (2008); Camp *et al.* (1998)]. Additionally the amino acids constituting the complementary strand[25] of the given genetic sequence should also be encoded. Thus the complementary strand should likewise be translated into three reading frames. From the above explanation it is clear that six reading frames should be translated from each genetic sequence in order to encode all the possible peptide (amino acid) sequences within a given genetic sequence.

- **tblastn:** compares peptide sequence queries to nucleotide sequence databases;
- **tblastx:** compares nucleotide sequence queries dynamically translated in all 6 of the reading frames to nucleotide sequence databases dynamically translated in all 6 reading frames as explained in the preceding item blastx. The six different reading frames created for the query sequence should all be compared to the six different reading frames created for each database entry thus for one query sequence and one database entry 6×6 pairwise comparisons of reading frames exist [Gish (2009)].

There are various other support programs in the WU BLAST package [Gish (2009)]. However, it is beyond the scope of this dissertation to investigate these programs.

[25]The sequence of the four nitrogenous bases or nucleotides (Adenine (A), Cytosine (C), Guanine (G) and Thymine (T)) mirror each other, in each strand of the double helix mirror, in a predefined manner: Adenine on the one strand always binds with Thymine on the other, and Cytosine always binds with Guanine [Bergeron (2003)]. Therefore, the sequence `ATTGCA` will occur as `TGCAAT` on the complementary strand of the helix [Paces (2001)].

2.5.3.4 RepeatMasker

RepeatMasker is a program that searches DNA sequences for interspersed repeats (Section 2.2) and low complexity DNA sequences[26] [Smit *et al.* (2003)]. The output of RepeatMasker is a detailed annotation of the repeats that are present in the query sequence, together with a modified version of the query sequence in which all the annotated repeats have been masked. Sequence comparisons by RepeatMasker is performed by the program *cross_match. Cross_match* is an efficient implementation of the Smith-Waterman-Gotoh algorithm developed by Green [Smit *et al.* (2003)].

RepeatMasker effectively masks simple repeats to avoid spurious matches in database searches, but is not written to detect and indicate all possible polymorphic simple repeat sequences [Smit *et al.* (2003)].

It is clear that, although RepeatMasker may detect some TRs, it is not a method devoted to the detection of TRs in general, nor, in particular, to the detection of microsatellites. For this reason, RepeatMasker will not be evaluated as an competing algorithm.

2.5.4 Concluding Remarks

The algorithm proposed in this dissertation, FireμSat, is specifically intended for the detection of microsatellites. For this reason Sputnik, (Section 2.5.1.1), TROLL (Section 2.5.1.2), Msatminer (Section 2.5.1.3) and STAR (Section 2.5.1.4) were included in the above literature overview.

However, it is apparent from our discussion that several algorithms detect TRs of any length, and thus indirectly detect microsatellites as a subset of these. Algorithms that we have discussed that belong to this category are ATRHunter (Section 2.5.2.1), FORRepeats (Section 2.5.2.2) and Tandem Repeats Finder (Section 2.5.2.3).

Other algorithms that deal with repeats were included namely, REPuter (Section 2.5.3.1) and RepeatFinder (Section 2.5.3.2).

Finally BLAST (Section 2.5.3.3) was included since, on the one hand, it is considered to be a classic algorithm in biological and computational terms. On the other it is also implemented by:

[26]Low complexity DNA sequences are sequences that contain biological information that is considered to be biologically less relevant. If the low complexity DNA sequences are masked, then the remaining sequence can be analyzed with software that is more expensive in terms of memory and runtime utilization [Van den Bergh (2006)].

- Msatminer (Section 2.5.1.3) to compare the occurrences of microsatellites of different genomes and by

- RepeatFinder (Section 2.5.3.2) to complete its classification of TRs.

RepeatMasker (Section 2.5.3.4) was included as an example of software that does not have the objective of detecting TRs but that has the ability to detect some of them.

There are numerous other software packages that also detect TRs, either directly or indirectly. It is possible to classify this software in various ways. Benson (1999) divided the algorithms that he investigated into three categories and mentions their shortcomings as follows:

- *Alignment algorithms*
  Alignment algorithms proposed by [Benson (1995), Kannan & Myers (1996) and Schmidt (1998)] have an excessive running time — their running time is exponential.

- *Algorithms from the field of data compression*
  An algorithm proposed by Milosavljevic & Jurka (1993) detects simple sequences. Simple sequences may or may not contain TRs. This algorithm makes no attempt to deduce a repeated pattern. Rivals *et al.* (1995) also developed an algorithm belonging to this category that is based on the presence of preselected patterns with $(1 \leq |motif| \leq 3)$. This algorithm suffers from severe limitations in terms of the motif length that is allowed, and in terms of the fact that the algorithm only searches for preselected motifs.

- *Algorithms that aim to find TRs more directly.*
  Of these algorithms, the one developed by Landau & Schmidt (1993) is limited by its definition of approximate repeats. The algorithm requires that two copies differ by k or fewer substitutions (Hamming distance) or by k or fewer substitutions and indels (unit cost edit distance). The requirement for a fixed number of differences rather than a percentage is regarded as unsatisfactory. Similarly, the treatment of substitutions and indels as equals, is regarded as unsatisfactory. The heuristic algorithm proposed by Karlin *et al.* (1988) is hampered in the same manner by the use of matching blocks separated by error blocks of fixed size. Sagot & Myers (1998) have proposed an exact algorithm that requires that the approximate pattern size and a range for the number of copies should be pre-specified. An earlier algorithm of Benson (1995) only finds TRs if they have a pattern size that is specified in advance.

Delgrange & Rivals (2004b) argue that an exact algorithm that entails the systematic detection of significant TRs in a way that is independent of the motif or of the sequence length is beyond the scope of present methods. In regard to existing software, Delgrange & Rivals (2004b) also distinguish between three different classes of algorithms and their shortcomings as follows:

- *Fast algorithms from the field of Computer Science.*
  In the field of Computer Science there are several fast algorithms that search for only two exact tandem repeats. Authors presenting these approaches include Main & Lorentz (1984); Kolpakov & Kucherov (1999) and Stoye & Gusfield (2002). Although these algorithms may be useful as filters to detect possible duplicate motifs they do not comply with the needs of molecular biologists [Delgrange & Rivals (2004b)].

- *Algorithms that do not make provision for the detection of TRs containing substitutions, deletions and insertions at once.*
  Algorithms in this category include those developed by Kolpakov & Kucherov (2001), as well as the algorithms developed by Landau *et al.* (2001) and Coward & Drablos (1998). These particular algorithms only make provision for substitutions.

- *Algorithms that detect TRs and allow for substitutions,insertions and deletions.*
  These algorithms include the work of Sagot & Myers (1998) who introduced a combinatorially exhaustive approach that identifies several possible motifs and alignments for each TR. The complexity of this approach depends exponentially on some parameters. The work of Rivals *et al.* (1997) is limited to small motifs and allows only indels between two of the motifs within a TR.

Wexler *et al.* (2005) also refer to the algorithms developed by Guan & Uberbacher (1996), Kolpakov & Kucherov (2001), Krishan & Tang (2004), Kurtz *et al.* (2001), Landau *et al.* (2001) and Rivals *et al.* (1995), noting that TRs detected by these algorithms are limited by constraints on their input data, search parameters, the type of mutations allowed and the number of such mutations. Wexler *et al.* (2005) claim that the time requirements of the algorithm proposed by Kannan & Myers (1996) is not suitable for the analysis of whole genomes that constitute millions of base pairs.

In spite of all the software mentioned and or discussed, it would appear that several researchers in the field of Molecular Biology do not use this software, but

instead, detect microsatellites by visually scanning genetic sequences[27]. Thus it seems that there is still a need for *effective* software to assist in the detection of microsatellites. From conversations with Dr. L.P. Wright at the University of Pretoria, published literature and developed software, it is clear that the effective detection of microsatellites confronts us with a twofold challenge. On the one hand there is the challenge of solving a computationally difficult problem within reasonable time and memory boundaries; and, on the other hand, there is the challenge of developing software that is *useable* from the perspective of the biologist community, especially in terms of output data.

In Section 2.6 criteria will be considered to serve as guidelines for the development of useable, computational tools to detect microsatellites.

2.6 Criteria for Tools to Detect Microsatellites

Below we have compiled a list of criteria that will contribute to the successful development of software tools for the detection of microsatellites. Our list constitutes criteria proposed by Benson (1999), a criterium suggested by Delgrange & Rivals (2004b) and two of our own criteria. Benson (1999) suggests the following criteria that should be pursued during the development of an effective (in terms of runtime and memory complexity) TR detection algorithm:

1. The avoidance of full scale alignment matrix computations in the case of alignment algorithms.
2. No *a priori* knowledge should be required pertaining to the pattern, pattern size or number of copies of the TR.
3. No restrictions should exist regarding the size of the repeats that can be detected.
4. Percentage differences between adjacent copies should be used and substitutions and indels should be treated separately.
5. A consensus pattern for the smallest repetitive unit in the TR should be determined.

Note in addition to Benson's criteria, the criterium set suggested by Delgrange & Rivals (2004b) is also endorsed, namely:

[27] In fact, the research carried out in this dissertation has been guided by feedback received from two such molecular biologists.

6. An exact algorithm should systematically detect significant TRs in a way that is independent of the motif.

Finally, in an attempt to contribute to the knowledge framework we suggest two of our own criteria:

7. An algorithm that detects microsatellites should be *flexible* in terms of penalties awarded to indels and mismatches.

8. Software to detect microsatellites should be *useable*, specifically in terms of output. By this we mean that analytically, biologically and statistically relevant output should be provided to the user. Furthermore, we suggest a hierarchical output that will enable the user easily to obtain the most relevant data.

2.6.1 Competing Algorithms

The two algorithms that we have chosen to evaluate as competitors to FireμSat are:

- Tandem Repeats Finder developed by Benson (1999) and,
- STAR: An algorithm to Search for Tandem Approximate Repeats, developed by Delgrange & Rivals (2004b).

Tandem Repeats Finder
Tandem Repeats Finder is a prominent algorithm throughout the literature on the detection of TRs. Both Delgrange & Rivals (2004b) and Wexler *et al.* (2005) compare the software implementations of their newly developed algorithms with Tandem Repeats Finder. Tandem Repeats Finder does not solely search for microsatellites and will detect TRs of length up to 2000 base pairs. Tandem Repeats Finder complies with all the criteria set by Benson (1999) but it is not an exact algorithm [Delgrange & Rivals (2004b)]. The output generated by Tandem Repeats Finder is of statistical and biological relevance and will be discussed in more detail in Chapter 3 and in Chapter 6. In Chapter 6 the emphasis will fall on the previously mentioned criteria compiled for developing computational tools to detect microsatellites effectively, as well as on the degree to which Tandem Repeats Finder succeeds to fulfill these criteria.

Examples of output generated by Tandem Repeats Finder are included on the CD that accompanies this dissertation.

STAR: an algorithm to Search for Tandem Approximate Repeats
STAR was developed to search specifically for microsatellites. Delgrange & Rivals (2004b) claim to have designed an exact algorithm. In their algorithm a PTR is obtained from the duplication of the motif (PTRE); an ATR can then be encoded as a number of duplications of the motif together with a list of mutations. Delgrange & Rivals (2004b) claim to have designed an algorithm that detects all significant TRs of a given motif, where significance is assessed by using the Minimum Description Length (MDL) criterion. Chapter 3, Section 3.2 provides more detail regarding the MDL criterion. At the time of the publishing of their paper, their algorithm required that one had to enter the motif whose TRs were to be detected. Thus STAR does not comply with the second of the criteria set for the development of computational tools to detect microsatellites effectively. However, Delgrange & Rivals (2004b) propose that STAR should be run with all possible Lyndon motifs[28] such that $2 \leq |motif| \leq 6$ in order to detect all TRs in a genetic substring. Chapter 3, Section 3.2.1 will elaborate on this issue.

Although STAR does not comply with all the criteria set by Benson (1999) and although the runtime of STAR is not as good as that of Tandem Repeats Finder (see Chapter 3), Delgrange & Rivals (2004b) can legitimately claim that STAR is an exact algorithm — a claim which Benson (1999) cannot make of Tandem Repeats Finder. Delgrange & Rivals (2004b) also claim that for the default values of Tandem Repeats Finder, STAR found more than 98 % of the TRs detected by Tandem Repeats Finder, whereas Tandem Repeats Finder detected at most 55% of the TRs detected by STAR. STAR also generates statistical, biological relevant data to which attention will be paid in Chapter 3, Section 3.2.2.

In Chapter 6, the focus will fall in more detail on the extent to which STAR fulfills the previously mentioned criteria that was compiled for the development of computational tools to detect microsatellites effectively. Examples of output generated by STAR are also included on the CD that accompanies this dissertation.

The proposed algorithm FireμSat
The algorithm, FireμSat, that is proposed in Chapter 4, was designed to comply to the criteria compiled for the development of computational tools to detect microsatellites effectively. Finite automata are applied, not alignment matrices. Thus item 1, proposed by Benson (1999) is not applicable. The only restriction is that our algorithm is only capable of detecting microsatellites, where $(2 \leq |motif| \leq 5)$. As Delgrange & Rivals (2004b) we can also claim that our

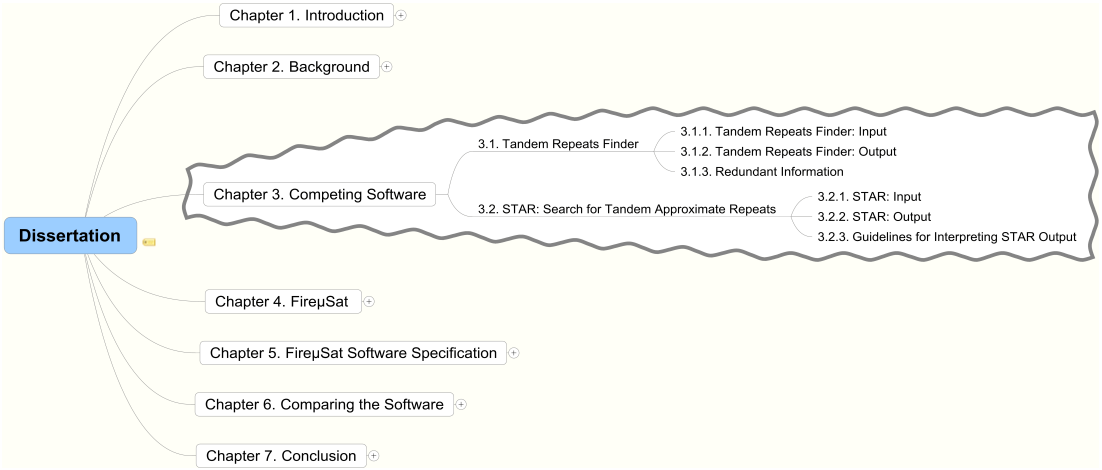
[28]The set of Lyndon motifs exclude motifs that are a rotation of another word (e.g. *tac* and *cta* while *act* remains in the set). The set of Lyndon words also exclude words that are made of the repetition of a shorter motif in that set (e.g. *atat* if *at* is an element of the set.)

algorithm is an exact algorithm, although we provide our user with the option of penalizing indels and substitutions separately. We aim thus to provide a lot of flexibility so that molecular biologists may adapt parameters according to their need. In contrast to the other algorithms, we will also provide the user with an hierarchical output of test results (in the sense that more relevant TRs may be investigated first) in an attempt to ease the task of the molecular biologist. In Chapter 4 more detail pertaining to our algorithm is given. The input and output of FireμSat are discussed in Chapter 5. FireμSat is, to the best of our knowledge, the first of its kind. Thus far, we have only encountered one direct attempt to detect approximate tandem repeats that occur on genomic sequences by means of finite automata, namely FORRepeats [Lefebvre *et al.* (2003)]. FORrepeats has been discussed in Section 2.5. In Chapter 6 FireμSat will be compared with Tandem Repeats Finder and STAR in reference to the previously mentioned criteria compiled for the development of computational tools to detect microsatellites effectively. Examples of output generated by FireμSat are included on the CD that accompanies this dissertation.

Before we introduce the theoretical foundations of FireμSat in Chapter 4 we describe, in Chapter 3, the selected competing algorithms, Tandem Repeats Finder and STAR, in terms of their input and output.

Chapter 3

Competing Software



In Chapter 2, a literature overview of algorithms that aim to detect tandem repeats on DNA was provided. Two algorithms that deal in an effective manner with the detection of microsatellites (as defined in Section 2.2) on DNA were identified namely, Tandem Repeats Finder [Benson (1999)] and STAR (Search for Tandem Approximate Repeats) [Delgrange & Rivals (2004b)]. "Effective" in the context of this dissertation, firstly implies algorithms that detect microsatellites containing substitutions, deletions, as well as, insertions at once. It also implies that the relevant algorithms have a running time smaller or equal to $O(np + nlog(n))$, where n is the length of the genetic sequence file under investigation and p is the length of the motif to be detected. Similar to FireμSat (the proposed algorithm) STAR detects microsatellites only, whereas Tandem Repeats Finder detects microsatellites as well as minisatellites (defined in Section 2.2) and satellites (defined in Section 2.2).

The required input to Tandem Repeats Finder (Section 3.1) and to STAR (Section 3.2) is not trivial. In each case, a variety of parameters are required, and it may not be clear to a first time user precisely what these mean and how they should be chosen. The output of these algorithms is similarly complex. For these reasons, it was decided to devote this current chapter to describing the input and output of these two algorithms. This description will also convey some sense of the nature of the algorithms in each respective case, without fully enumerating the algorithmic details, the latter being regarded as outside the scope of this present study.

In Chapter 4 the theoretical background, as well as the specifications of the algorithms proposed in this dissertation, FireμSat, will be presented. In Chapter 5 the input and output of FireμSat are discussed. Chapters 3, 4 and 5 thus provide a basis for Chapter 6, in which a comparison is made between Tandem Repeats Finder, STAR and FireμSat in relation to the criteria presented in Section 2.6. At that stage, the three algorithms will also be evaluated in terms of their run-time performance.

3.1 Tandem Repeats Finder

Tandem Repeats Finder was developed by Gary Benson, an associate professor in the Department of Biology and Computer Science at the Boston University [Benson (2005a)]. Tandem Repeats Finder was published in 1999. At the time of writing, the 2003 (fourth) version of Tandem Repeats Finder was the latest online available version. This version provides the user with four different downloadable applications. The options provided to the user are as follows: Windows 9x/NT/Me/2000/XP; Windows/Linux Command Line versions; Linux (Graphical GTK version); and a Mac OS X version [Benson (2005b)].

Tandem Repeats Finder models tandem repeats by percent identity and frequency of insertions and deletions (indels) between adjacent motif repeats. The program uses statistically based recognition criteria. Tandem Repeats Finder consists of detection and analysis components. The detection component uses a set of statistically based criteria to determine *candidate* tandem repeats. The analysis component aims to produce an alignment for each candidate. If the analysis component is successful then it gathers a number of statistics regarding the alignment and the nucleotide sequence [Benson (2003b)]. Tandem Repeats Finder detects microsatellites ($|motif| < 6$) as well as repeats with long motif lengths [Benson (1999)]. The time complexity of Tandem Repeats Finder is linear in the sequence length.

The program can execute without specifying a motif or a motif length. However, Tandem Repeats Finder is not an exact algorithm according to Delgrange & Ri-

vals (2004b). In Section 3.1.1 the input of Tandem Repeats Finder is investigated and in Section 3.1.2 the output of the program. Both of these sections discuss the information presented in Benson (2003b) and Benson (2003c).

3.1.1 Tandem Repeats Finder: Input

Tandem Repeats Finder is installed as *trf*. After typing *trf* at the command line the following output appears:

```
Please use: trf File Match Mismatch Delta PM PI MinScore MaxPeriod
```

Each word after *trf* in this message indicates an input parameter required by Tandem Repeats Finder. Each of these parameters is now discussed in turn.

1. `File`: *The input DNA sequence file in FASTA format.*
   The DNA sequence file contains the genomic sequence where intra-genomic comparison, in terms of the detection of tandem repeats, should take place [Benson (2003c)]. FASTA format is discussed in Section 2.3. Multiple genomic sequences in the same file are acceptable as long as the data is in FASTA format. The benefit thereof is that it enables the user to process a large sequence file consisting of multiple genomic sequences without creating very small files for each genetic sequence. The genetic data retrieved by Wright and reported on in Wright *et al.* (2007) (discussed in Section 2.4), is a practical example of data where the processing of multiple genomic sequences in one file is beneficial and where there should be reported on each new header of each short genomic sequence too. This type of data is in contrast to the data of a whole genome, e.g. the fusarium genome.

2. `Match`, `Mismatch`, `Delta`: *Alignment parameters that represent the weights for matches, mismatches and indels respectively.*
   These parameters are used for Smith-Waterman style local alignment wraparound, dynamic programming [Benson (2003b)]. Dynamic programming is a very general algorithmic optimization technique. It is applicable when a large search space can be structured into a succession of stages, in such a manner that:

   - the initial stage consists of trivial solutions to sub-problems,
   - it is possible to calculate each partial solution in a later stage by recurring a fixed number of partial solutions in an earlier stage and,
   - the final state contains the overall solution [Brown (1995)].

Dynamic programming is implemented by the Smith-Watermeyer algorithm in such a manner that it takes alignments of any length, at any location, in any sequence, and determines whether an optimal alignment[1] can be found. Scores or weights are assigned to each character-to-character comparison: positive for exact matches and substitutions; and negative for indels [Brown (2004)]. Dynamic programming determines the optimal alignment in relation to these weights.

Lower weights entered as the alignment parameters of Tandem Repeats Finder allow alignments with more mismatches and indels. `Match` = 2 has proven effective with `Mismatch` and `Delta` ranging between 3 and 7. Mismatch and indel weights are interpreted as negative numbers. The values allowed are 3, 5 and 7. In these types of alignment options, 3 is more permissive and 7 is less permissive (i.e. $-3 > -7$) [Benson (2003b)]. Benson (2003c) recommends the values 2, 7 and 7 for `Match`, `Mismatch` and `Delta`, respectively.

3. `PM` *and* `PI`: *Detection Parameters*
   Detection parameters consist of a matching probability Pm and an indel probability Pi. $Pm = 0.8$ and $Pi = 0.1$ by default and cannot be altered in the revised, 2003 version of the program.

   Pm is directly related to $P(heads)$ in the interpretation of a specific alignment as a Bernoulli sequence and represents the average percentage identity between the aligned copies. This is explained more fully in 4 below.

   Pi represents the indel probability — i.e. the average percentage of insertions and deletions between the copies allowed.

   These detection parameters serve as a type of extremal bound — i.e. as a quantitative description of the most divergent copies to be detected.

   Thus, adjacent repeats (approximate or exact) of a motif will contain some matching characters in corresponding positions and possibly some non-matching characters. The proportion of matches and the proportion of indels, are constrained by the fixed values of Pm and Pi respectively [Benson (1999)].

   The UNIX version of Tandem Repeats Finder provides probabilistic data for `PM` values of 80 and 75 and `PI` values of 10 and 20. The UNIX version documentation, however, mentions that the best performance can be

[1] The optimal alignment of two sequences implies the determination of an alignment that will obtain the highest score (the score is calculated by a function as specified by the developer) from aligning two given sequences in all possible manners [Van den Bergh (2006)].

achieved with values of `PM` = 80 and `PI` = 10. By using the values of `PM` = 75 and `PI` = 20 Tandem Repeats Finder obtains results very similar to those calculated when `PM` = 80 and `PI` = 10 but the processing time may be 10 times longer [Benson (2003c)] .

4. `Minscore`: *Minimum alignment score*
   The minimum alignment score indicates the alignment score that must be met or that must be exceeded for a tandem repeat to be reported [Benson (2003b)]. The alignment of two or more possibly approximate tandem repeats of a motif (referred to as a pattern by Benson) of which the length of the repeated motif is n, is modelled by a sequence of n-independent Bernoulli trials[2]. Below is an interpretation of a particular alignment as a Bernoulli sequence. In this alignment, the motif is `TTC`, with motif length 3. The top row represents a substring of a genetic sequence constituting 14 nucleotides. The second row represents exact copies of the PTRE, `TTC`. Benson refers to the second row as the *consensus sequence*. The bottom row is the Bernoulli sequence that is construed as an *interpretation* of the alignment. Each head (H) in the Bernoulli sequence is interpreted as a match between aligned nucleotides. Each tail (T) represents a mismatch, insertion or deletion [Benson (1999)]. In the example below the top row represents the actual genetic string; the middle row represents a PTR where the PTRE or motif is equal to TTC; and the bottom row represents the outcome of the Bernouli trials.

T	T	C	T	T	C	T	T	C	C	T	C	T	T
T	T	C	T	T	C	T	T	C	T	T	C	T	T
H	H	H	H	H	H	H	H	H	T	H	H	H	H

   In the example there are $4\frac{2}{3}$ repetitions of the motif `TTC`, and one mismatch — a `T` is replaced by a `C`. `Minscore` can be used to ensure that a certain minimum number of repeats of motifs in tandem must occur before qualifying for being reported. For example, suppose that the researcher only requires a report on tandem repeats constituting 5 or more repeats of the identified motif in tandem. Then one can set `Match` = 2 and `Minscore` = 30. This weight and alignment score will result in the required number of repeats, because if there is perfect alignment of a motif with length 3 and if at least 5 repeats occur, then there will be at least 15 exact matches, each with a weight of 2. Since $15 \times 2 = 30$, five successive perfect repeats will be reported, but not a TR consisting of only 3 successive perfect repeats.

[2]Bernoulli trials are associated with a succession of coin tosses. The probability of heads (or a match, or a success) is a fixed value over the tosses or trials. In this sense, the tosses / trials are independent of each other.

5. `Maxperiod`: *Maximum period size*
   Benson (1999) defines period size as the most common distance between corresponding characters in the alignment. In Tandem Repeats Finder, period size is the program's best guess at the length of the TREs that are detected within an identified TR. Note that in exceptional cases, the length of the PTRE, (termed the consensus pattern by Benson) may differ from the period size. As a default, Tandem Repeats Finder will find all TRs that constitute concatenated motif repeats with a period size between 1 and 2000.

   However, the period size can be made smaller in length by setting `Maxperiod` [Benson (2003b)].

Tandem Repeats Finder provides three additional options (`-f`, `-m` and `-d`), that the user may specify as part of the command line input. These "switches" are included to provide the user with additional options in terms of output. They are merely added to the input sequence in the command line. Whether a particular user will choose to utilize these "switches" will depend on the nature of the analysis of the molecular biological problem at stake. A description of each of these additional switches follows:

1. `-f` *Flanking sequence*
   The flanking sequence of a tandem repeat consists of the 500 nucleotides on each side of the repeat. If this switch is set, then the output file records the flanking sequence. Information regarding the flanking size may be useful for PCR[3] primer[4] detection.

2. `-m`: *Masked sequence file*
   If this switch is set then a masked sequence file is generated. The masked sequence file is in FASTA format (discussed in Section 2.3) and contains a copy of the genetic sequence but every character contained within a TR is converted to the letter 'N'. (The masking of repeats and the reason for masking repeats have been discussed in Section 2.5.3.4.) The string `> masked` is appended to the end of the sequence description line.

[3]PCR is the abbreviation for polymerase chain reaction, a molecular biological technique for amplifying or copying a selected region of a DNA molecule, so that its sequence is multiplied many times in a laboratory [Kahn (2005)].

[4]A primer is an oligonucleotide (a short, single stranded DNA molecule synthesized chemically under automated conditions generally 15 - 50 nucleotides in length) which is complementary to a specific region within a DNA or a RNA molecule. Primers are used to initiate synthesis of a new strand of complementary DNA at that specific site, in a reaction or series of reactions catalyzed by a *DNA polymerase* (an enzyme which catalyzes the addition of a nucleotide to a nucleic acid molecule) [Lemon & Barbour (1993)].

3. `-d`: *Data file*
   The data file is a text file, containing the same information in the exact same order, as the summary table file of repeats (discussed as part of the output in Section 3.1.2), plus the detected motif, as well as consensus sequences - a consensus sequence is a concatenation of motifs (PTREs) or in Benson's terminology consensus patterns. An example of a consensus sequence and TRs referred to as repeat sequences by Benson is included below. The top row represents the actual genetic sequence; the bottom row represents a PTR generated by Tandem Repeat Finder to which the detected TR within the genetic score can be compared.

   T T C T T C T T C C T C T T
   T T C T T C T T C T T C T T

   The data file does not contain any labeling and is thus suitable for additional processing, with, for example, a perl script, external to the program.

   Below is an example of a data file obtained after the sequence BL157 has been run by me. BL157 is data of the **Cylindrocladium Pauciramosum** (Section 2.4):

```
>BL157
GACAGACAGACAGACAGGATCAGAACGGGCGGCAATGTGTTGTTCCTAAG
AATGGTTAGAATGAGACATCTAAGCAAAGCTATTTCATGACTCACTTTAG
TTTTCGCATGTAACTGGGCTTGTGCAGGAGATGGTAGGATTGGCCGTAGA
CATTATCAAAGAACACTTCTGCAAGATGTTCCTGTACATCCTTTGGGGGC
AAGGACTCAGAACCCTCGTGCAGTAACTTGTTTTCTTCTTCCTCTTGAAC
TTGAAGGGAATTTGGTCTATCATCGCCAACAATCTTCGGTTTTGAAGGAC
CTCTGGCCCAGGCCTCAAGATCCGGACCGAAAGCTTCATCAGCACCGCGC
TTCTTGGAACCACCCTTTGTTGTTGTTGTTG
```

3.1.2 Tandem Repeats Finder: Output

Tandem Repeats Finder generates a summary table of repeats as well as an alignment explanation,s as output.

1. *The summary table.*
   The summary table provides the following information:

   - Indices of the detected TR relative to the start of the sequence.
   - The period size of the TR. This is the most common matching distance between corresponding characters in the alignment and usually corresponds to the motif length.

- The number of repeats aligned with repeats of the consensus motif. Consider, for example, the following:

  ```
  TTC TTC TTC CTC TT
  TTC TTC TTC TTC TT
  ```

  The top row constitutes a substring of a genetic sequence and the bottom row consists of copies of the detected consensus motif or PTRE. In this case the number of repeats aligned with repeats of the consensus motif is 4.7.

- The length of the consensus motif (PTRE) (If the consensus motif is `TTC` then the length of the consensus motif is 3.) This may differ slightly from the period size.

- The overall percentage of matches between adjacent repeats in the TR. Consider the tandem repeat: `TTCTTCTTCCTCTT`. The detected PTRE or motif is `TTC` and the overall percentage of matches may be calculated as 81, as shown below.
  Let $TRE_1 =$ `TTC`, $TRE_2 =$ `TTC`, $TRE_3 =$ `TTC`, $TRE_4 =$ `CTC` and $TRE_5 =$ `TT` and denote the three elements of TRE_1 as $TRE_{1,1}$, $TRE_{1,2}$ and $TRE_{1,3}$. Thus $TRE_{1,1} =$ `T`, $TRE_{1,2} =$ `T` and $TRE_{1,3} =$ `C`. Using the same notational conventions for $TRE_2 \cdots TRE_5$, the overall percentage of matches may be calculated as follows:
  $TRE_{1,1} = TRE_{2,1}$
  $TRE_{1,2} = TRE_{2,2}$
  $TRE_{1,3} = TRE_{2,3}$
  $TRE_{2,1} = TRE_{3,1}$
  $TRE_{2,2} = TRE_{3,2}$
  $TRE_{2,3} = TRE_{3,3}$
  $TRE_{3,1} \neq TRE_{4,1}$
  $TRE_{3,2} = TRE_{4,2}$
  $TRE_{3,3} = TRE_{4,3}$
  $TRE_{4,1} \neq TRE_{5,1}$
  $TRE_{4,2} = TRE_{5,2}$
  The above gives 9 matches and 2 mismatches — $9 \div 11 \times 100 = 81,818181 \approx 81$.

- The alignment score. If the weight assigned to each matching character is equal to 2 and if there are x matching characters in the alignment then the alignment score will be $2x$. The alignment score refers to matches of the consensus sequence[5] and the relevant TR, detected in the genetic sequence.

[5]The consensus sequence is a sequence of adjacent PTREs, repeated as many times as there are TREs in a string under consideration.

Tabular Explanation of Sequence: BL157

Indices	Period Size	Copy Number	Consensus Size	Percent Matches	Percent Indels	Score	A	C	G	T	Entropy (0 - 2)
233 – 246	3	4.7	3	81	0	19	0	35	0	64	0.94
367 – 381	3	5.0	3	100	0	30	0	0	33	66	0.92

Table 3.1: TRF Summary File Output

- Percentage of composition of the four nucleotides (adenine (`A`), cytosine (`C`), guanine (`G`) and thiamine (`T`)).
- The measure of entropy[6] based on percent composition.

An example of a summary table created by TRF after run on BL157 is provided in Table 3.1.

This summary table that is generated by Tandem Repeats Finder thus provides an overview of detected TRs. In order to obtain more detailed information about a specific TR, the user can click on the applicable index range. This will result in an alignment explanation of the index range, as explained below.

2. *The alignment explanation.*
The alignment explanation includes the actual genetic sequence stretching throughout the TR (e.g. from indices 233 - 246 in the example below). It corresponds to- and elaborates on the information presented in the above-described summary table that was created by Tandem Repeats Finder.

   The consensus sequence is provided below the TR.

   Thus, multiple repeats of the detected PTRE are concatenated to each other in the case of microsatellites.

   To assist in the clarification of the additional information to be further discussed below, two examples of alignment explanations of detected TRs are given.

[6]The entropy estimation of a DNA sequence provides a measure of its complexity and randomness level [Vinga & Almeida (2004)].

Example 1: Alignment explanation BL157, base pairs 233 - 246 Section 2.4

```
    Indices: 233--246  Score: 19
    Period size: 3  Copynumber: 4.7  Consensus size: 3

        223 GTAACTTGTT

                       *
        233 TTC TTC TTC CTC TT
          1 TTC TTC TTC TTC TT

        247 GAACTTGAAG


Statistics Matches: 9,  Mismatches: 2, Indels: 0
        0.82            0.18        0.00

Matches are distributed among these distances:
   3    9  1.00

ACGTcount: A:0.00, C:0.36, G:0.00, T:0.64


Consensus pattern (3 bp): TTC

Found at i:375 original size:3 final size:3
```

Example 2: Alignment explanation fusarium genome, base pairs 6471 - 6483 (Section 2.4)

```
    Indices: 6471--6483  Score: 19
    Period size: 2  Copynumber: 7.0  Consensus size: 2

       6461 TTTTAATGAC


       6471 TA TA TA TA -A TA TA
          1 TA TA TA TA TA TA TA

       6484 GGGAAAAAAG
```

```
Statistics Matches: 10,  Mismatches: 0, Indels: 2
        0.83            0.00        0.17

Matches are distributed among these distances:
   1    1  0.10
   2    9  0.90

ACGTcount: A:0.54, C:0.00, G:0.00, T:0.46


Consensus pattern (2 bp): TA

Found at i:6537 original size:1 final size:1
```

Additional information that accompanies the alignment explanation is as follows:

- The 10 base pairs before and after a TR are shown by default. In the case of example 1, `GTAACTTGTT` and `GAACTTGAAG` are the 10 base pairs that occur before and after the detected TR, respectively. However, as mentioned above, if the user so wishes, then 500 characters of flanking sequence on each side of the TR can be shown by adding the character `-f` to the input sequence.
- The symbol * appears above the pair of aligned lines to indicate a mismatch. In example 1 a `T` has been replaced by a `C`. Thus, a * appears above the `C` of the detected TR.
- The symbol - appears within the applicable sequence to indicate an insertion or a deletion. If a deletion occurred then - will appear within the *detected* TR. If an insertion occurred then - will appear within the *consensus sequence*. In example 2 a deletion has occurred. Thus a - appears within the detected TR, replacing the absent `T`.
- Statistics pertaining to matches, mismatches, insertions and deletions accompany the alignment explanation. These statistics refer to overall matches, mismatches, insertions and deletions between adjacent repeats (TREs) *within* the sequence and not *between* the sequence and the consensus sequence (concatenation of PTREs). The calculation of these statistics is done in a similar manner as described in Section 3.1.2. These statistics are included in both example 1 and example 2.

- Distances between matching characters at corresponding positions are listed in the following order: distance (number of nucleotides between matching nucleotides plus 1), number at that distance (number of matching or corresponding nucleotides using the above- mentioned distance) and percentage of all matches. Consider the data generated in example 1:

```
Matches are distributed among these distances:
   3     9   1.00
```

Here 3 represents the distance. In the first example, 9 matches occur - all the matching nucleotides have a matching distance of 3. Therefore the percentage of all matches occurring at distance 3 is 100% output as 1.00 by Tandem Repeats Finder.

Consider the output of the second example. Here two distances are provided in the distance description as shown below:

```
Matches are distributed among these distances:
   1     1   0.10
   2     9   0.90
```

To clarify the meaning of the above distance description, a copy of the TR under consideration is reproduced below (in the second line), as well as the consensus sequence (in the third line). The indices of the genomic sequence are given on the first line.

```
6471 6472 6473 6474 6475 6476 6477 6478 6479 6480 6481 6482 6483
 T    A    T    A    T    A    T    A   -A    T    A    T    A
 T    A    T    A    T    A    T    A   TA    T    A    T    A
```

If there is referred to the indices where the base pairs occur then it is possible to explain the number of nucleotides at a distance and the percentage of all matches as follows:

Distance 1 represents the two consecutive `A`'s occurring at index position 6478 and index position 6479. There is only 1 occurrence of matching nucleotides with a distance of 1. This is indicated by the second 1 in the top row of the distance distribution output.

The 0.10 in the top row of the distance distribution output indicates this fact and represents 10%.

For distance 2 in the distance distribution output, matches are as follows:

Index 6471 (`T`) matches index 6473 (`T`).
Index 6472 (`A`) matches index 6474 (`A`).

Index 6473 (T) matches index 6475 (T).
Index 6474 (A) matches index 6476 (A).
Index 6475 (A) matches index 6477 (A).
Index 6476 (A) matches index 6478 (A).
Index 6480 (T) matches index 6482 (T).
Index 6481 (A) matches index 6483 (A).

Thus 9 matches occur at distance 2 — hence the 9 in the second row of distance distribution output. In total throughout the detected TR ranging from index 6471 to index 6483, 10 matches occurred, 9 of these matches have a matching distance of 2. This explains the entry of 0.9 (i.e. of 90%) as the third entry of the second row of the distance distribution output.

- The ACGT count reflects the percentage of each nucleotide (adenine (A), cytosine (C), guanine (G) and thiamine (T)) in the TR under investigation.
- The motif, termed the consensus pattern by Benson that constitutes 3 nucleotides or base pairs (bp abbreviated) [Benson (2003b)].
- There is a little additional output provided as well, but not of importance for the current discussion.

3. *Additional files.*
   Finally, recall that the user has the option to specify whether Tandem Repeats Finder should generate a masked sequence **-m** file and a data file **-d**. The formats of these files are discussed in Section 3.1.1.

3.1.3 Redundant Information

It should also be noted that Tandem Repeats Finder detects repeats of motif lengths ranging from 1 to 2000 nucleotides. Thus, if a TR contains numerous repeats then the same repeat will be detected at various period sizes (motif lengths). However, this is restricted to the three best scoring motif lengths. Consider for example a TR consisting of 30 TR-elements with period size 3. Then the TR will most likely also be detected at motif length (period size) 6 with 15 TR-elements and motif length (period size) 9 with 7.5 TR-elements. It is also possible that the same period size or motif length may be detected more than once with different scores and slightly different instances [Benson (2003a)].

3.2 STAR: Search for Tandem Approximate Repeats

Delgrange & Rivals (2004b) introduced STAR formally (by means of a peer-reviewed publication) in 2004. One of its authors, Delgrange, is in the Department of Theoretical Computer Science at the Wallonie-Academy of Brussels [Delgrange (2005)]. The other, Rivals, is a researcher in Computer Science and Bioinformatics at the CNRS (Centre National de la Recherche Scientifique), where he is a member of the *Methods and Algorithms for Bioinformatics* team.

STAR is an algorithm that detects "significant" TRs of a selected motif in a DNA sequence in FASTA format. *Significance* is assessed in terms of a measure that has to do with the extent to which substrings can be subjected to compression. Although full details are beyond the scope of the current discussion, a little more is said about this matter below.

STAR searches a genetic sequence for motifs entered by the user. For a selected motif, it returns a description of all the TRs of that motif, as well as optimal alignments of the various detected TRs and the selected motif [Delgrange & Rivals (2004b)].

The package uses the so-called Minimum Description Length (MDL criterion) to distinguish between *significant* TRs and *random* TRs. The MDL Principle was originally proposed by J. Rissanen in 1978 as a computable approximation of the so-called Kolmogorov complexity. Both the MDL Principle and Kolmogorov complexity are two well researched topics. An overview of these two topics will be provided but the details of these are beyond the scope of this dissertation.

The MDL is based on the insight that any regularity in data can be used to compress the data i.e. to describe the data using fewer symbols than the number of symbols needed to describe the data literally. The general choice of researchers in the field for carrying out the compression is to write a program in a so-called general-purpose computer language such as C, C++ or PASCAL.

The choice leads to the definition of Kolmogorov complexity. The Kolmogorov complexity of a sequence s, denoted by $K(s)$, is defined as the shortest program (for a fixed computer) that prints s. This definition is invariant within a constant if one alters the universal Turing Machine or the programming language. Therefore, $K(s)$ is considered as the absolute measure of the information content of s. However, it can be shown that there does not exist a computer program that, for every set of data s, when given s as input it will return the shortest program that outputs s.Therefore, Kolmogorov complexity is not computable [Grunwald *et al.* (2001)]. But every compression algorithm provides an approximation of $K(s)$: the length $K_C s$ of the compressed sequence of s by a compression algorithm C is an upper bound of $K(s)$ and can therefore, be considered as a tentative measure

of the information content of s. (If the compression algorithm C is improved, then the measure $K_C s$ converges to $K(s)$) [Rivals *et al.* (1995)].

In the next subsections, STAR will be discussed in a similar manner to the discussion of Tandem Repeats Finder.

3.2.1 STAR: Input

In order to be able to run STAR the user has to have access to either a Windows or a Linux system. A Linux operating system has been used to run the data presented on the CD that accompanies this dissertation. At the command line STAR.linux should be entered. The following will appear on the screen:

```
-i SeqFile -m Motif | -M MotifFile
          [-na -po PositionOffset -help]
SeqFile: file containing the sequence in Fasta, Genbank
          or Embl format
Motif: the motif to search for is a string over
          alphabet [ACGT]
MotifFile: a file with one motif per line, each motif
          is searched independently, this option
          excludes option -m
-na: option without the output of alignments of
          tandem repeats;
default is with alignments
-po PositionOffset: set a position offset that is
          added to output positions;
```

Thus, STAR prompts the user for input in terms of the following parameters and output control options:

1. *Parameter options.*

   - A DNA sequence file (`-i SeqFile`) in FASTA format (Section 2.3) has to be specified. It will be searched for significant tandem repeats. The DNA sequence file may be in upper case or lower case. Only strings over the alphabet $\Sigma = \{A, C, G, T\}$ are allowed in the genetic sequence input file. In contrast to Tandem Repeats Finder, STAR does not allow multiple genomic sequences in the same file. Thus, if the genetic data to be examined consists of several short genetic sequences, then a separate file has to be created for each of these genetic sequences. Unfortunately, this process is very time consuming

for the user, especially if data similar to that of Wright *et al.* (2007) has been generated.

- A DNA motif (`-m Motif`) over the alphabet $\Sigma = \{A, C, G, T\}$ is required. The motif may for example, be `TTG`. The motif or PTRE may be a microsatellite of any length in upper case or lower case. All zones containing the selected motif will be identified by STAR.
- The DNA motif file (`-M MotifFile`) is an alternative for the `-m Motif` option. The DNA motif file consists of various motifs arranged one per line. STAR is intended to execute successively on each one of them, one at a time [Delgrange & Rivals (2004a)].

2. *Output control options.*

   - The user may select whether alignments should be included in the output file or not. The output control option (`na`) appears on the screen. If the user enters n next to (`na`) then alignments will be omitted from the output. However, alignments are created by default as part of the output [Delgrange & Rivals (2004a)].
   - A position offset (`-po PositionOffset`) is entered as an integer and is added to the positions of the TR base pairs found by STAR in order to determine the exact stretch of the detected TR. An entered sequence is often a portion of, for example, a chromosome. Therefore, to obtain the position of the TR in your sequence relative to the complete chromosome, it is necessary to enter the offset position of the sequence in the chromosome. If the position offset option is activated then STAR adds the offset to all positions before printing. The default value of the position offset is zero [Delgrange & Rivals (2004a)].

3.2.2 STAR: Output

If only one motif was entered to search for, then the output of STAR is one output file. If the input option `-M MotifFile` has been selected, then various files are produced. For each motif, if the genetic sequence contains at least one tandem repeat of that motif, then the results are written in a file named $SeqFile.motif$.

Consider for example the file BL153.seq used as DNA sequence input file. If the entered motif to search for is `CCA` and the motif is detected within BL153.seq, then an output file BL153.CCA will be created. If multiple zones of the same motif are found within the genetic sequence file, then all the different zones will be listed, the first as ZONE 1, the second as ZONE 2, etc. in the order of their appearance in the genetic sequence file. The different zones are thus all listed in

the same output file.

The authors claim that the output provides detailed information pertaining to each zone in a structured manner that enhances postprocessing of the results. This claim will be contested in Chapter 6. The output file can also easily be parsed [Delgrange & Rivals (2004a)].

To further clarify the output format of files generated by STAR, examples of the contents of two output files are provided below. The first example output file, BL141.ttg, was generated from a file, BL141, containing one of the sequences generated by Wright *et al.* (2007). The second example output file, BL141ins.cca, was generated after the original input file, BL141, had been modified to BL141ins.

```
Genetic input file: BL141.txt

>BL141
CCACCACCACCACCAACACAATTGCACCGCTAGTGGCTATATTTGATGCC
CTCAAAATTCCCGCACCGTGGGCACCAGAGGCCAAGGATTTCGACTACGC
AAACACGACTTTGCTGATTATGGGTGGTGGATCCAGCACCGGCAAATTTG
GCGTACAATTAGCCAAGTTAGCAGGCATTGGCAAGATTGTTGTTGTTGTT
G
```

Example 1, output file: BL141.ttg

```
ZONE      1 BEGIN_POS       187 END_POS       201 LG      15 GAIN     23
A      0 C      0 G      5 T     10 N      0 %AT  66 %GC  33 Biais GC 1.00
Phase 0
Consensus columns, counts of matches, substitutions, and deletions
Position     1    2    3
Nb_Match     5    5    5
Nb_Subst     0    0    0
Nb_Del       0    0    0
Insertion columns number: 0 and list of positions and counts
         Match  Sub  Del  Ins
Totals      15    0    0    0
Percents  100.0  0.0  0.0  0.0
Nb_Motifs  5.00 Percent_of_exact_motifs 100.00 Consensus 1 ttg
Pat            ttgttgttgttgttg
Seq       187  ttgttgttgttgttg

Nb_mutations 0
FILE BL141.txt LG 201 MOTIF_LG 3 MOTIF ttg NB_ZONES 1
```

Modified genetic input file: BL141ins

```
>BL141
CCACCACCTACCACCAACACAATTGCACCGCTAGTGGCTATATTTGATGC
CCTCAAAATTCCCGCACCGTGGGCACCAGAGGCCAAGGATTTCGACTACG
CAAACACGACTTTGCTGATTATGGGTGGTGGATCCAGCACCGGCAAATTT
GGCGTACAATTAGCCAAGTTAGCAGGCATTGGCAAGATTGTTGTTGTTGT
TG
```

Example 2, output file: BL141ins.cca

```
ZONE      1 BEGIN_POS         1 END_POS        20 LG      20 GAIN     18
A      7 C     12 G      0 T      1 N      0 %AT  40 %GC  60 Biais GC 1.00
Phase 0
Consensus columns, counts of matches, substitutions, and deletions
Position     1    2    3
Nb_Match     6    6    6
Nb_Subst     1    0    0
Nb_Del       0    0    0
Insertion columns number: 1 and list of positions and counts
Position     2
Nb_Ins       1
         Match  Sub  Del  Ins
Totals      18    1    0    1
Percents  90.0  5.0  0.0  5.0
Nb_Motifs  6.33 Percent_of_exact_motifs 68.42 Consensus 1 cca
Pat            ccaccacc-accaccaCcac
Seq         1  ccaccaccTaccaccaAcac
                       ^       ^

Nb_mutations 2
Mutations_list  9,t, 17,d,
FILE BL141ins.txt LG 202 MOTIF_LG 3 MOTIF cca NB_ZONES 1
```

The following information is distinguished for each detected TR:

1. *General information.*
   The general information is written in the first two lines of the output as follows:

   - *ZONE* (Line 1), All the TRs listed in the same output file have the same consensus motif (identified PTRE). Thus the zone number indicates whether it is the first, second, third, etc. occurrence of a TR with the specific consensus motif, within the genetic sequence file.
   - *BEGIN_POS* (Line 1), the start position of a particular TR.
   - *END_POS* (Line 1), the end position of a particular TR.
   - *LG* (Line 1), the length of the detected TR.
   - *GAIN* (Line 1), reflects the local compression gain.
   - `A`, the number of occurrences of adenine.
   - `C`, the number of occurrences of cytosine.
   - `G`, the number of occurrences of guanine.
   - `T`, the number of occurrences of thiamine.
   - `N`, the number of occurrences of undetermined nucleotides.
   - %`AT`, percentage of `AT` (`AT` represents the percentage of weak hydrogen bonds [Strachan & Read (2004)]).
   - %`GC`, percentage of `GC` (`GC` represents the percentage of strong hydrogen bonds [Strachan & Read (2004)]).
   - *Biais* `GC`, representing the `GC` bias.
   - *Phase*, indicates the pattern phase at the beginning of the TR. A TR may start at any position of the motif. For example, if the motif is `TTG` then *phase 0* = `T`; *phase 1* = `T` and *phase 2* = `G`. The *Phase* contributes towards the positioning of the alignment [Delgrange & Rivals (2004a)].

2. *Consensus description.*
   STAR computes alignment in such a manner that it is clear where the position of a new copy of a motif starts. The TR is divided into successive repeats. STAR then counts, for each position in the motif, the nucleotides that occur at those positions respectively. The consensus description of STAR in relation to a detected TR and an identified motif consists of the following output:

   - *A table in which the columns represent the respective positions in the motif.*
     The entry in the first row (*Nb_Match*) at a given column (i) is the number of exact matches for motif position i. In the first example

(BL141.ttg), 5 matches occurred in each of the motif positions 1, 2 and 3. Similarly, the second and third rows contain the number of mismatches (substitutions) (*Nb_Subst*) and deletions (*Nb_Del*) in each of the motif positions, respectively. Note that there are neither deletions nor mismatches in the first example(BL141.ttg).

The values of *Nb_Match*, *Nb_Subst* and *Nb_Del* are calculated relative to a position i within the motif, where $1 \leq i \leq |motif|$. Thus, if the computed consensus motif is, for example `TTG`, then *Position 1* = `T`, *Position 2* = `T` and *Position 3* = `G`. Thus *Nb_Match* = 5 for *Position 1*, if there are five occurrences of `T` in *Position 1* throughout the detected TR. Similarly, if a `C` is in *Position 1* and is the only mismatch (substitution) in *Position 1*, then *Nb_Subst* = 1 for *Position 1*. Note, there is one mismatch in the second example (BL141ins.cca). The mismatch occurred in the first position and is indicated as such.

- *The number of insertion columns detected within the specific TR.*
  If there were 4 insertion positions within a consensus alignment of a TR, then STAR would output:*Insertion column number: 4 and list of positions and counts*. Consider the output file BL141ins.cca. One insertion was detected by STAR. Thus STAR output *Insertion column number: 1 and list of positions and counts*. The *list of positions and counts* is discussed in the next item below.

- *The list of positions and counts*
  consists of a table in which column entries indicate where insertions occurred, in relation to the consensus motif for the TR under exploration. Consider a motif $u = u_1u_2 \cdots u_p$.

  If an insertion is detected between u_i and u_{i+1}, then this is indicated by allocating the number u_i to the value `Position`. Underneath position is a row *Nb_Ins*, if there have occurred X insertions between the u_i and u_{i+1}, then X is output next to *Nb_Ins*.

  As evident in the output file BL141.ttg shown above, no insertions occurred in the sequence BL141. However, in the second example data BL141ins.txt, there was one insertion and it occurred after the second character of the third repeat TRE relative to the start of the detected TR, so that *Position* = 2 and *Nb_Ins* = *1*.

3. *Distribution of matches, substitutions and indels.*
   The 4 columns of the table that represent the distribution of matches, substitutions, deletions and insertions contain the headings *Match*, *Sub*, *Del* and *Ins* respectively. The entries in the first row, labeled *Totals*, indicate the exact number of matching, substituted, deleted or inserted nucleotides

that were detected. The entries in the second row, labeled *Percents*, provide the same information in percentage terms.

In the case of the output BL141.ttg file (example 1) shown above, no mismatches (substitutions), deletions or insertions occurred. Therefore, *Match* has been allocated the output value 100.0.

Consider the output file, BL141.ins.cca, of example 2. For the detected PTRE = `cca`, the following is given:
Match = 18, *Sub = 1*, *Del = 0*, *Ins = 1*, *Percents Match = 90.0*, *Percents Sub = 5.0*, *Percents Del = 0.0* and *Percents Ins = 5.0*.

4. *Number of repeats and consensus pattern.*
An output line that contains the following information is provided:

   - *Nb_motifs*: the number of motif repeats that occurred in a specific zone. In the case of the first output example (BL141.ttg) of STAR, STAR counted 5 repeats. In the case of BL141ins.cca STAR counted 6.33 repeats. We note that strictly speaking, *Nb_motifs* should be a rational number (in the present example $6\frac{1}{3}$), should be output rather than the real number (6.33) that is provided.
   - *Percent_of_exact _motifs*: The ratio between the number of exact matches and the number of mutations is provided. In the case of the example (BL141.ttg), the ratio of exact matches is shown to be 100%.
   - *Consensus*: This serves as a boolean flag that indicates whether a particular motif entered as parameter, was detected or not. In the former cases *Consensus* = 1, else *Consensus* = 0. The motif under consideration is output alongside the consensus flag. Note that in some cases STAR does not detect a TR of the motif entered as parameter. Instead, it may find a TR that has a similar motif (pattern).
     In the first example, the motif that was entered was `ttg`. STAR has detected a TR where repeats of `ttg` occur. Therefore *Consensus* = 1 is printed out. In the second example the motif `cca` was entered, STAR detected a TR where TREs of `cca` occurred.

5. *Alignment.*

The detected TR is aligned against a concatenation of consensus motifs (concatenated repeats of the detected PTRE — i.e. a PTR). An example of the described alignment is included below.

```
Pat ccaccacc-accaccaCcac
Seq ccaccaccTaccaccaAcac
                ^       ^
```

The alignment is computed by Wrap Around Dynamic Programming (Section 3.2). The output is written in four separate lines for each TR. There is a maximum of 60 characters in each line. The first output line is labeled as *Pat*. This line contains concatenated copies of the consensus motif. The second output line is labeled as *Seq*. *Seq* contains the actual detected TR. In the third line, each mutation is marked with a $\wedge$. Furthermore, irrespective of whether the input format of the genetic sequence was upper or lower case, upper case is used to indicate mutations in *Pat* and *Seq*, while all the other nucleotides are output in lowercase. The fourth line is empty.

6. *Mutations list.*
   The mutation list consists of the list of mutations that occurred within a TR. The first line of output contains the total number of mutations (*Nb_mutations*) that occurred within the detected TR. In the case of the first output example, BL141.ttg, the number of mutations is 0 while the number of mutations of the second example, BL141ins.cca, is 2.

   Delgrange & Rivals (2004a) developed single symbol codes to represent the entire spectrum of possible mutations. These codes are provided in Tables 3.2, 3.3 and 3.4.

   The codes represented in the tables are used to list the occurrences of the different mutations that were identified within the detected TR. The list of mutations are preceded by the word *Mutations_list*. Each entry within the mutation list consists of a numerical value, followed by an alphabetical value, separated by a comma. The numeric value represents the relative position of the mutation in the detected TR pertaining to the alignment. The alphabetical character represents the one of the mutations listed in the tables above. STAR outputs $9, t$ to indicate the insertion of t at index position 9 relative to the start of the detected TR in the second example, BL141ins.cca. Also for example 2 STAR outputs $17, d$ to indicate the replacement of c with a at index position 17 relative to the start of the detected TR.

   The final line provides the sequence filename and length, the length of the detected motif, the motif itself, and the total number of detected TRs containing the particular motif. This final line starts with the word FILE.

3.2.3 Guidelines for Interpreting STAR Output

1. *The absence of an output file.*
   If the execution of STAR terminates properly, but an output file has not been created, then it implies that a TR of the selected or very similar to the selective motif has not been found.

Substitution of	Substitution by	Code
a	c	a
a	g	b
a	t	c
c	a	d
c	g	e
c	t	f
g	a	g
g	c	h
g	t	i
t	a	j
t	c	k
t	g	l
n	a	u
n	c	u
n	g	u
n	t	u
a	n	v
c	n	v
g	n	v
t	n	v

Table 3.2: Single symbol codes for substitutions

Deletion of	Code
a	m
c	n
g	o
t	p
n	w

Table 3.3: Single symbol codes for deletions

Insertion of	Code
a	q
c	r
g	s
t	t
n	x

Table 3.4: Single symbol codes for insertions

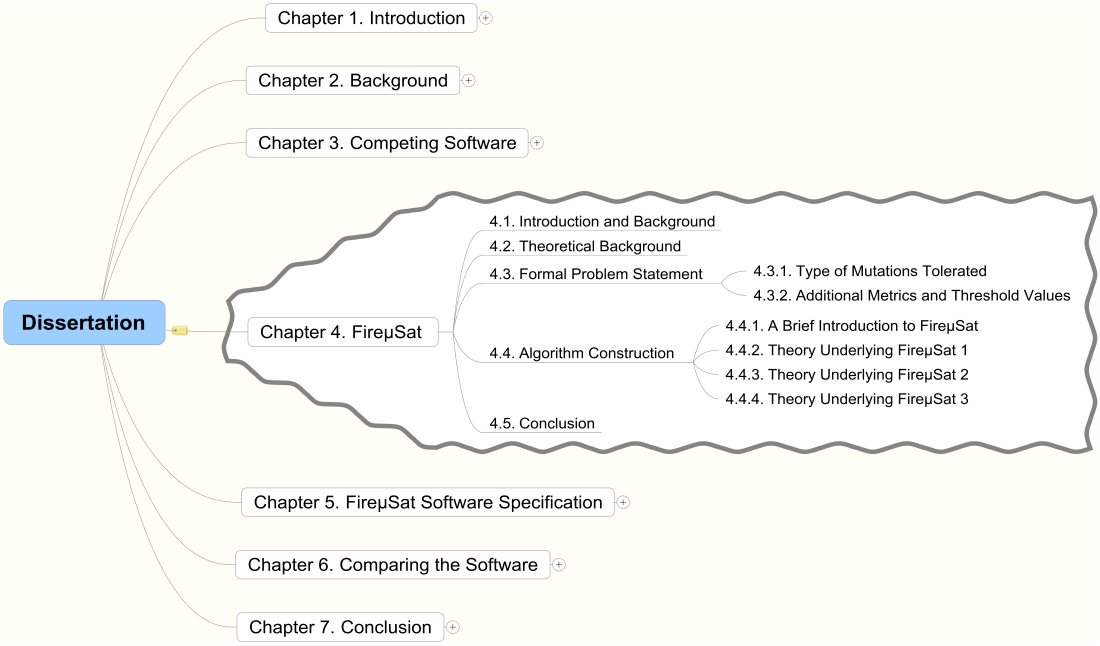
2. *Consensus pattern differs from the motif input parameter.*
   In this case the detected TR is a TR of a motif very similar to the motif used as input parameter, or the detected TR is a complex TR, not obviously corresponding to the motif, consisting of several patterns. These types of TRs usually suggest other potential patterns for the search.

3. *Redundancy.*
   The file that is output by STAR reports on one or more detected TRs associated with a given motif. It is possible that another run of STAR on the same input sequence, but using a slightly different motif, would produce TRs that overlap with those in the first run. In the sense that the same TR may be reported on twice, there is some redundant data generated. Delgrange & Rivals (2004b) mentions that such ambiguity is inherent to the tandem repeats search problem. Earlier in this Chapter (Section 3.1.2) it was noted that Tandem Repeats Finder suffers from a similar type of drawback.

The required input to Tandem Repeats Finder (Section 3.1) and to STAR (Section 3.2) has been discussed. In Chapter 4, the theoretical background as well as the specifications of the algorithms proposed in this dissertation, FireμSat, will be presented. In Chapter 5 the input and output of FireμSat are discussed in a similar manner to the discussion of that of Tandem Repeats Finder and STAR in this Chapter. Chapter 6 is a comparison of the three different algorithms: Tandem Repeats Finder, STAR and FireμSat at the hand of the criteria proposed in Chapter 2 (Section 2.6).

Chapter 4

FireμSat



4.1 Introduction and Background

FireμSat inspired three algorithmic implementations for TR detection that rely on finite automata theory and that produce similar output. Each implementation has a slightly different theoretical foundation with regards to the manner in which finite automata have been implemented. The 3 implementations — FireμSat$_1$, FireμSat$_2$ and FireμSat$_3$ realise the same specification. However, it is possible to distinguish between three layers of abstraction — FA theory forms the basis for the specifications, which are realised in the three implementations.

The actual implementation of the different FireμSat versions were undertaken by:

- A.P.F. Marais (FireμSat$_1$), and
- T.R. Fourie (FireμSat$_2$) under the supervision of the author, in partial fulfilment of their respective B.Sc. (Hons) degrees at UNISA.
- Additionally P.V. Reyneke implemented FireμSat$_3$ in accordance with the author's specifications.

The details of the algorithm's input requirements and of its output are presented in Chapter 5, in a similar manner to the implementation details of Tandem Repeats Finder and STAR in Chapter 3. The input parameters of FireμSat$_1$ and FireμSat$_2$ are similar. FireμSat$_3$ has only been implemented to search for microsatellites with a motif length of three. The objective of the implementation of FireμSat$_3$ is threefold:

- Firstly it is of theoretical and practical relevance in the sense that it shows that it is possible to implement FireμSat$_3$.
- Secondly the opportunity is provided to compare the data generated by FireμSat$_3$ to the data generated by FireμSat$_1$ and FireμSat$_2$. The generated output data is equivalent. Note, FireμSat$_3$ only reports any detected TR once, whereas FireμSat$_1$ and FireμSat$_2$ generate duplicate data that can be eliminated during post-processing.
- Thirdly the runtime of FireμSat$_1$ and FireμSat$_2$ can be compared to the runtime of FireμSat$_3$.

The results of the three implementations (Tandem Repeats Finder, STAR and FireμSat$_2$) are compared in Chapter 6 in terms of the criteria presented in Chapter 2, Section 2.6. As mentioned in Chapter 1, a brief comparison between FireμSat$_2$ and IMEx [Mudunuri & Nagarajaram (2007)] will also be included.

The remainder of this Chapter is laid out as follows:

- Section 4.2 explains the automata technology used in the development of FireμSat.
- Section 4.3 defines the problem of detecting microsatellites in genetic substrings in a formal manner that facilitates the subsequent explanation of FireμSat.
- In Section 4.4 three outlines are provided of how automata technology can be used to detect TRs in a DNA string, culminating in pseudo-code for the three different FireμSat algorithms.
- Section 4.5 concludes this chapter.

4.2 Theoretical Background

FireμSat$_1$, FireμSat$_2$ and FireμSat$_3$ are DFA-based algorithms. It therefore seems appropriate to review relevant automata terminology before exploring the construction of the three FireμSat-algorithms. Cohen (1997) defines an FA informally as a collection of three things:

1. a finite set of states of which one is designated to be the start state (the initial state), and some (possibly none) designated final states;
2. a finite alphabet of possible input letters; and
3. a finite set of transitions that indicates for each state and for each letter of the input alphabet which state to go to next.

More formally an FA is defined as:

1. A finite set of states $Q = \{q_0, q_1, q_2...q_n\}$, of which q_0 is designated to be the start state.
2. A subset of Q called the final states (F).
3. A finite alphabet $\Sigma = \{x_1, x_2, x_3...x_s\}$. In our case $\Sigma = \{A, C, G, T\}$.
4. A transition function $\delta : Q \times \Sigma \rightarrow Q$, which is total and complete [Hopcroft & Ullman (1979) and Cohen (1997)].

A string or word from the alphabet that matches a path from the start state to a final state is said to be accepted by the FA. If $\Sigma = \{A, C, G, T\}$, then it is possible to draw a deterministic FA that accepts only the word `ACG` as in Figure 4.1.

This FA called, say FA_{ACG}, can also formally be defined as follows:
$FA_{ACG} = (Q, \Sigma, \delta, q_0, F)$ where:
$Q = \{q_1, q_2, q_3, q_4\}$
$\Sigma = \{A, C, G, T\}$
$F = \{q_3\}$, and δ is shown below:

States	A	C	G	T
q_0^-	q_1	q_4	q_4	q_4
q_1	q_4	q_2	q_4	q_4
q_2	q_4	q_4	q_3^+	q_4
q_3^+	q_4	q_4	q_4	q_4
q_4	q_4	q_4	q_4	q_4

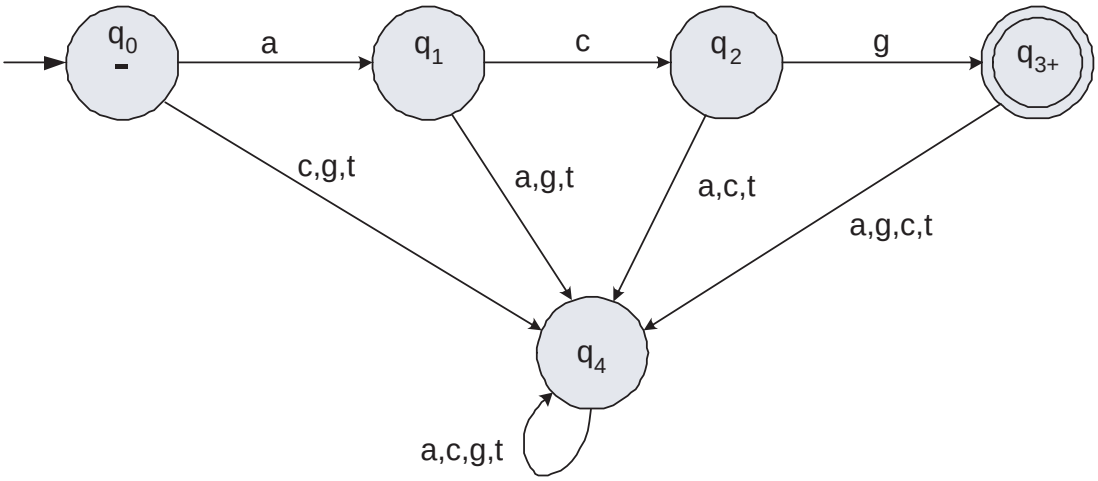


Figure 4.1: An FA accepting the PTRE or the motif `ACG`

It should be noted that this FA is also referred to as a deterministic FA (DFA), where "→" and "-" are used to indicate the start state, while double circles and "+" are used to indicate final states. These conventions will be followed in the remainder of this dissertation.

A state such as q_4 is called a *sink state*, because for all the possible input characters from the alphabet $\Sigma = \{A, C, G, T\}$ it is impossible to leave state q_4 — all the transitions loop back into state q_4. An FA can therefore have transitions which do not lead to strings being accepted, and can have states which do not play a role in accepting strings. In fact, all the edges leading to the sink state do not contribute to the acceptance of strings, and sink states play no role in accepting strings.

The sole function of FAs are to accept certain input strings and to reject other input strings. Consequently, some people also call FAs finite acceptors. Languages accepted by FAs are regular languages [Cohen (1997)]. Regular languages can be defined by regular expressions. Regular expressions can more formally be defined by the following rules [Cohen (1997)]:

- Rule 1: Every word consisting of a single letter that belongs to the alphabet Σ is a regular expression. Also Λ, which represents the empty word or the word with no length, is a regular expression.
- Rule 2: If r_1 and r_2 are regular expressions, then so are the following regular expressions:
    1. $r_1 r_2$ where $r_1 r_2$ is the concatenation of the regular expression r_1 to the regular expression r_2 in this order ($r_1 r_2$ is not equivalent to $r_2 r_1$).

2. $r_1 + r_2$ where $r_1 + r_2$ is the sum of the two regular expressions.
3. $(r_1)^*$, the Kleene closure of r_1. If $r_1 = \mathtt{A}$ then $(r_1)^* = \{\Lambda, \mathtt{A}, \mathtt{AA}, \mathtt{AAA}, \mathtt{AAAA}, \cdots\}$. $(r_1)^*$ is thus the set of each word in r_1 concatenated n times with itself and the other words in r_1 as well as the words constructed in $(r_1)^*$, where $0 \leq n < \infty$ and where Λ is taken to be the result of concatenating a word 0 times with itself. Note, $(r_1)^*$ is equivalent to r_1^*.

- Rule 3: Nothing else is a regular expression.

Languages associated with regular expressions are regular languages. Regular languages are type 3 languages of the Chomsky hierarchy of grammars.[1] The following rules define languages that are associated with regular expressions:

- Rule 1: The language associated with the regular expression that consists only of a single letter of the alphabet, is the set consisting of that one letter alone. The language associated with Λ is the set $\{\Lambda\}$, a one-word language.
- Rule 2: Let r_1 be a regular expression associated with the language L_1 and let r_2 be a regular expression associated with the language L_2, then:
  1. The regular expression $r_1 r_2$ is associated with the product, $L_1 L_2$ language. Thus, if $L_1 = \{\mathtt{AC}\}$ and $L_2 = \{\mathtt{G}\}$ then $L_3 = L_1 L_2 = \{\mathtt{ACG}\}$. Notice that in this case L_1, L_2 and L_3 are all one word languages. If $r_1 = \mathtt{A}(\mathtt{A})^*$ and $r_2 = \mathtt{C}(\mathtt{C})^*$ then $L_1 L_2 = \{\mathtt{AC}, \mathtt{AAC}, \mathtt{ACC}, \mathtt{AACC}, \mathtt{AAAC}, \mathtt{AACC}, \mathtt{ACCC}, \cdots\}$.
  2. The regular expression $r_1 + r_2$ is associated with the union of the sets $L_1 + L_2$. If $L_1 = \{\mathtt{AC}, \mathtt{GT}, \mathtt{TT}\}$ and $L_2 = \{\mathtt{AA}, \mathtt{CC}, \mathtt{GT}, \mathtt{TA}\}$, then $L_1 + L_2 = \{\mathtt{AA}, \mathtt{AC}, \mathtt{CC}, \mathtt{GT}, \mathtt{TA}, \mathtt{TT}\}$. If $r_1 = (\mathtt{A})^*$ and $r_2 = (\mathtt{C})^*$, then $L_1 + L_2 = \{\Lambda, \mathtt{A}, \mathtt{C}, \mathtt{AA}, \mathtt{CC}, \mathtt{AAA}, \mathtt{CCC}, \cdots\}$.
  3. The language that is associated with the regular expression r_1^* is L_1^* — i.e. the Kleene closure of the set of words, L_1. If $L_1 = \{\mathtt{A}\}$ then $L_1^* = \{\Lambda, \mathtt{A}, \mathtt{AA}, \mathtt{AAA}, \cdots\}$.

From these rules it is clear that there is a regular language associated with every regular expression. FAs are acceptors of regular languages. Kleene proved in 1956 that any language that can be defined by a regular expression can also be defined by an FA [Cohen (1997)]. It is beyond the scope of this dissertation to go into the details of the proof; details thereof can be found in Cohen (1997). However, attention will be paid to *Part 3, Rule 2* of the proof, because

[1] Noam Chomsky, a linguist, gave several mathematical models for languages [Cohen (1997)].

it provides the systematic way of constructing FAs that was used in Section 4.4.2.

Part 3 of Kleene's theorem states:

> "Every language that can be defined by a regular expression can also be defined by an FA."

Rule 2 of part 3 of Kleene's theorem states:

> "If there is an FA called FA_1 that accepts the language defined by the regular expression r_1 and there is an FA called FA_2 that accepts the language defined by the regular expression r_2, then there is an FA that shall be called FA_3 that accepts the language defined by the regular expression $r_1 + r_2$."

The proof that FA_3 exists can be given in terms of a constructive algorithm. The general principles underlying the construction of FA_3 are as follows: Starting with two machines FA_1 consisting of states $x_1, x_2, x_3, \cdots$ and FA_2 consisting of states $y_1, y_2, y_3, \cdots$, build a new machine FA_3 with states $z_1, z_2, z_3, \cdots$. Each state z_i in FA_3 can be viewed as combining two states: one from FA_1, say x_j, and another from FA_2, say y_k. This will be expressed by saying that z_i is "x_j or y_k".

The start state of FA_3 is then "x_{start} or y_{start}". If either the x part or the y part of the z state is a final state in its respective FA, then the z state is a final state of FA_3.

In moving from one z state to another when reading a letter from the input string, we investigate what happens to the x and y parts of z and go to the corresponding new z part. Letting δ_1, δ_2 and δ_3 denote the transition functions of FA_1, FA_2 and FA_3, it is possible to express the relationship as follows:

> Suppose $z_i = x_j$ or y_k. Suppose too that $\delta_1(x_j, p) = x_{new}$, $\delta_2(y_k, p) = y_{new}$ and $\delta_3(z_i, p) = z_{new}$. Then:
> $z_{new} = x_{new}$ or y_{new}.

As there are only finitely many x's and finitely many y's, there can be only finitely many z's. Note that not every combination of x and y state necessarily produces a reachable z state in FA_3 — a z state will only be defined in FA_3 if it is possible for a string over the relevant alphabet to exist in such a manner, that the particular state can be reached if the string is input. (For a z state to be included within an FA generated by FireμSat there must exist a path that leads from the start state to the corresponding x and y state.)

States	A	C	G	T
$z_1 = x_1$ or y_1	x_2 or $y_2 = z_2$	x_4 or $y_5 = z_3$	x_4 or $y_5 = z_3$	x_4 or $y_5 = z_3$
$z_2 = x_2$ or y_2	x_4 or $y_5 = z_3$	x_3^+ or $y_3 = z_4^+$	x_4 or $y_5 = z_3$	x_4 or $y_5 = z_3$
$z_3 = x_4$ or y_5	x_4 or $y_5 = z_3$	x_4 or $y_5 = z_3$	x_4 or $y_5 = z_3$	x_4 or $y_5 = z_3$
$z_4^+ = x_3^+$ or y_3	x_4 or $y_5 = z_3$	x_4 or $y_5 = z_3$	x_4 or $y_4^+ = z_5^+$	x_4 or $y_5 = z_3$
$z_5^+ = x_4$ or y_4^+	x_4 or $y_5 = z_3$	x_4 or $y_5 = z_3$	x_4 or $y_5 = z_3$	x_4 or $y_5 = z_3$

Table 4.1: Transition table, FA_3

In this manner it is possible to construct a machine that can accept the sum of two regular expressions if the machines that accept the component regular expressions are already separately known to us.
(*End of proof by means of constructive algorithm,*[Cohen (1997)].)

To illustrate briefly the construction of $FA_3 = FA_1 + FA_2$ consider the following example:

Let FA_1 be the FA that accepts only the word ACG, represented in Figure 4.2(a), and FA_2 be the FA that accepts only the word AC, represented in Figure 4.2(b).

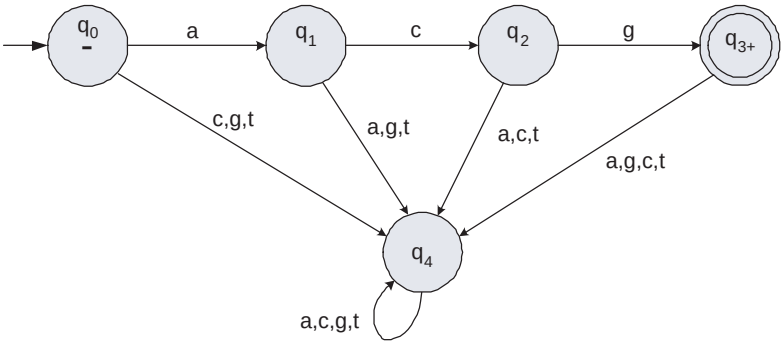
FA_3 is to accept the language $L = \{\texttt{AC}, \texttt{ACG}\}$. The pictorial presentation of FA_3 accepting L is represented in Figure 4.2.

FA_1 and FA_2 can be used to construct the transition table of FA_3 that accepts L. More formally $FA_3 = (Q, \Sigma, \delta, q_0, F)$, where:
$Q = \{z_1, z_2, z_3, z_4, z_5\}$
$\Sigma = \{A, C, G, T\}$
$F = \{z_4, z_5\}$ and
δ is represented by the transition table of FA_3, Figure 4.1.

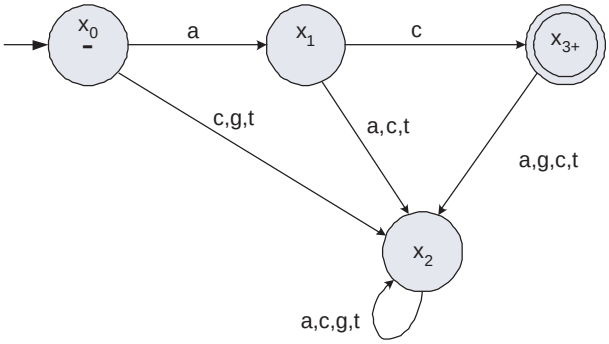
Suppose that AC is input to FA_3. Then, from the table it follows that $\delta(z_1, A) = z_2$ and $\delta(z_2, C) = z_4^+$. As a consequence, the conclusion is drawn that AC is in the language defined by FA_1+FA_2. Using functional notation somewhat more loosely, the foregoing is expressed by the following sequence of equalities:
$\delta(z_1, AC) = \delta(\delta(z_1, A), C) = \delta(z_2, C) = z_4^+$.

It can be seen from the above transition table that one is able to determine whether a final state of FA_3 is designated as final, because it includes a final state of FA_1, because it includes a final state of FA_2 or because it includes final states of both FA_1 and FA_2.

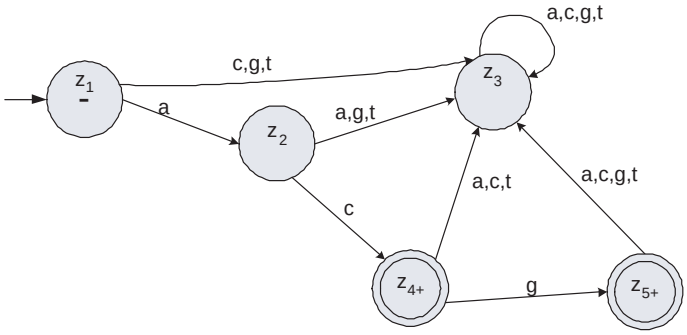
Thus, z_4 is a final state of FA_3, because it includes x_3^+ which is a final state of FA_1. On the other hand z_5^+ is a final state of FA_3, because it includes y_4 which is a final state of FA_2. In general, a final state of FA_3 could include final states of both FA_1 and FA_2. However, there is no such final FA_3 state in the



(a) FA_1 — An FA accepting `ACG`



(b) FA_2 — An FA accepting `AC`



(c) $FA_3 = FA_1 + FA_2$

Figure 4.2: An Illustration of FA addition — $FA_3 = FA_1 + FA_2$

current example. Note that state z_3 is a sink state. For all the input letters of the alphabet $\Sigma = \{A, C, G, T\}$ FA_3 will remain in state z_3.

In general, it is always apparent which "old" final states are included in a "newly" constructed finite state — i.e. which "old" final state(s) cause the "new" final state to be final. In principle it is possible to make a partition between the respective final states of FA_3 based on the origin of the component final states of the original FAs — FA_1 and FA_2 in this case.

Exactly the same principle would hold, *pari passu*, if more than two FAs were added together. This observation is important for the development of FireμSat.

However, there are also FAs that do more than simply accept a regular language. Certain FAs generate output. In this regard Mealy and Moore machines are to be distinguished.

Hopcroft & Ullman (1979) defines a Mealy machine more formally as a six tuple $Me = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where Q, Σ, δ and q_0 are defined similarly to the formal definition of the FA. Δ is the output alphabet and λ maps $Q \times \Sigma$ to Δ , thus $\lambda(q, a)$ gives the output associated with the transition from state q on input a. The output of Me in response to input $a_1, a_2, \cdots, a_n$ is $\lambda(q_0, a_1), \lambda(q_1, a_2), \cdots, \lambda(q_{n-1}, a_n)$, where $q_0, q_1, \cdots, q_n$ are states such that $\delta(q_{i-1}, a_i) = q_i$ for $1 \leq i \leq n$. If the input sequence is of length n, then the output sequence is also of length n.

The algorithm, FORRepeats discussed in Section 2.5, implements FAs that originate from the Mealy paradigm.

In contrast to Mealy machines whose output is determined by the arc followed when making a transition from one state to the next, Moore machines print output only in relation to the state that is reached — irrespective of which arc was followed to get to that state. Cohen (1997) defines a Moore machine as follows:

1. A finite set of states $Q = \{q_0, q_1, q_2, \cdots q_n\}$ where q_0 is determined to be the start state.

2. A finite alphabet of letters $\Sigma = \{a, b...\}$ for the construction of input strings.

3. A finite alphabet of output characters $\Delta = \{x, y, z...\}$.

4. A transition table that shows for *each* state and *each* input letter, which state will be reached next.

5. An output table that displays what character from Δ is printed by each state as it is entered.

It is also possible to define a Moore machine in a more formal manner as a six tuple: $Mo = \{Q, \Sigma, \Delta, \delta, \lambda, q_0\}$ where: Q, Σ, δ, and q_0 are the same as in the FA. Δ is the output alphabet and, λ is a mapping from each state in Q to Δ, representing the output associated with each state. The output of Mo in response to input $a_1, a_2, \cdots, a_n, n \geq 0$, is $\lambda(q_0), \lambda(q_1) \cdots \lambda(q_n)$, where $q_0, q_1, \cdots, q_n$ represent the sequence of states such that $\delta(q_{i-1}, a_i) = q_i$ for $1 \leq i \leq n$. Notice that the Moore machine will give output of length $n + 1$ if n is the length of the input sequence [Hopcroft & Ullman (1979)].

The input alphabet Σ need not be the same as the output alphabet Δ. Notice that a Moore machine does not define a language of accepted words as an FA does. The notion of a set of final states is not defined.

Thus, every possible string of a Moore machine creates an output string. The reading process is terminated when the last input letter is read and the last output letter is printed. There are, however, several ways to convert Moore machines into language-definers. The pictorial representations of Moore machines are very similar to those of FAs. Instead of only having the name of the state inside the circle representing the state, the output character printed by the state is also specified.

The two symbols inside the state (circle) are separated by a "\". On the left side of the "\" is the name of the state and on the right side is the output from that state.

Following the less formal conventions of Cohen (1997), we can represent a Moore machine that will determine exactly how many times the substring `ACG` occurs in a genetic input string as follows:

Input alphabet: $\Sigma = \{A, C, G, T\}$
Output alphabet: $\Delta = \{0, 1\}$
Names of states: q_0, q_1, q_2, q_3 where q_0 is the start state.

Moore Machine Transition Table: counting `ACG` occurrences

Old State	Output by old State	New State after input `A`	New State after input `C`	New State after input `G`	New State after input `T`
q_0	0	q_1	q_0	q_0	q_0
q_1	0	q_1	q_2	q_0	q_0
q_2	0	q_1	q_0	q_3	q_0
q_3	1	q_1	q_0	q_0	q_0

The graphical representation of the above Moore Machine is as represented in Figure 4.3.

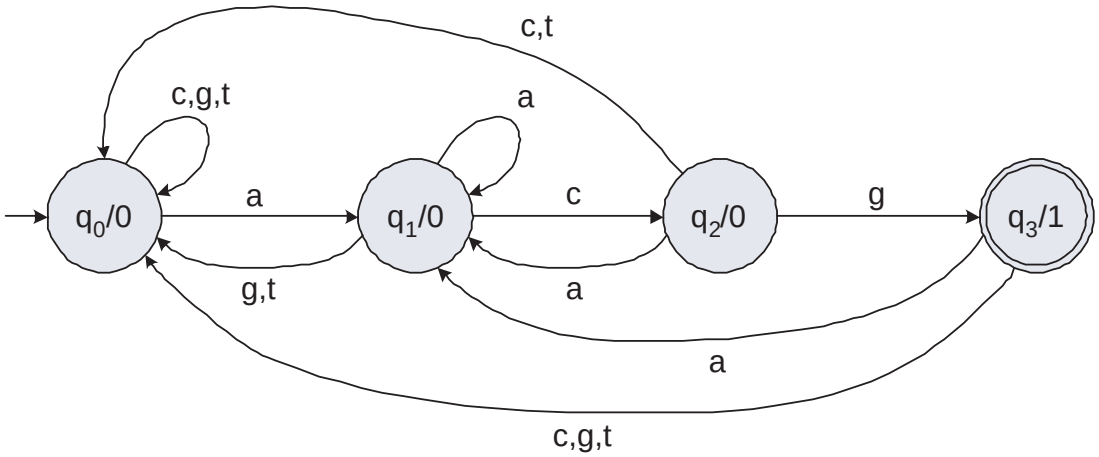


Figure 4.3: A Moore machine printing a 1 for each ACG substring

It is clear that every state of this machine outputs the character 0, except for state q_3 which outputs 1. The only way of reaching state q_3 is by reading an ACG substring. For example, consider the input string CGTTTACGACGT. The number of ACG substrings is equal to the number of 1's in the output string, which is shown in the last row below:

Sequence: CGTTTACGACGT

Input		C	G	T	T	T	A	C	G	A	C	G	T
State	q_0	q_0	q_0	q_0	q_0	q_0	q_1	q_2	q_3	q_1	q_2	q_3	q_0
Output	0	0	0	0	0	0	0	0	1	0	0	1	0

Now that the relevant theoretical concepts of FAs have been surveyed, the algorithmic development of FireμSat will be discussed in terms of this theory. The problem statement will be formally stated in a way that facilitates the explanation of the algorithm FireμSat in the next section.

4.3 Formal Problem Statement

Frequent use will be made of the core terminology introduced in Chapter 2. Therefore some of the relevant definitions will be recapitulated briefly.

In Chapter 2 a TR in a genomic sequence, is defined as a string of nucleotides that is characterized by a certain motif that introduces the string followed by at least two "copies" of the motif. If the motif ACG with motif length 3 ($|motif| = 3$) is considered, then ACGACGACGACGACG is a TR. Exact copies of the motif (as in the case of the above example) are called PTRs; otherwise (in the case of the

inclusion of non-exact copies in the TR) they are referred to as ATRs. An ATR is thus a string of nucleotides repeated consecutively at least twice, with a limited number of small differences between the instances being tolerated. An example of an ATR is: `ACGACTACGACGAC`. In the absence of further qualification, reference to a TR should be construed as a reference to either a PTR or an ATR.

A TR element (TRE) that matches the identified motif of the TR will be referred to as a PTR element (PTRE). A TRE that does not match the motif is referred to as an ATR element (ATRE). We have also defined microsatellites to be TRs with a motif (PTRE) length such that $2 \leq |motif| \leq 5$. It has been made clear that the term microsatellite is used to refer to a TR that contains PTREs and that may also contain ATREs.

ATRs on genetic sequences are defined in terms of the following more formal conventions. A PTR whose motif ρ is repeated p times where $p \geq 1$, is denoted by ρ^p. An ATR u that is derived from this PTR ρ^p, must also have the motif (ρ) as its prefix. It therefore has the form $\rho u_2 \cdots u_p$ where each ATRE, $u_k (k = 2 \cdots p)$, is the result of at most ε mutations on ρ. Here ε is called the *motif error*. In theory, ε could be anywhere in the range $0 \leq \varepsilon \leq |\rho|$.

However, when running FireμSat, the user is required to choose a maximum value for ε that complies with certain practical considerations. In determining whether the string $\rho u_2 \cdots u_p$ is to be construed as an ATR, this value of ε represents the maximum number of mutations (or errors) that are tolerated in deciding whether or not, for each $k = 2 \cdots p$, u_k represents an acceptable ATRE. In the next section the types of mutations that are tolerated are discussed. Here it is emphasized that the following toleration limits on ε apply for a given ρ.

1. If $|\rho| = 2$ or $|\rho| = 3$ then only zero or one error is tolerated; i.e. ε may be chosen as either 0 or 1. (The default is 1.)

2. If $|\rho| = 4$ or $|\rho| = 5$ then zero, one or two errors are allowed, i.e. ε may be chosen as either 0, 1 or 2. (The default is 2.)

Recall that the interest is in detecting microsatellites, which means that $2 \leq |\rho| \leq 5$.

Consider an example where $\rho =$ `ACGTT`. Then $|\rho| = 5$ and the user may consequently select the maximum number of errors to be either 0 or 1 or 2. If the user selects "2", then `ACT` would be regarded as an ATRE, since it may be construed as the motif in which two deletions (see Section 4.3.1) have occurred. Likewise `ACGT` could be regarded as an ATRE, since it may be seen as the motif in which one deletion has occurred. (See Section 4.3.1.) However, `AC` will not be regarded as an ATRE.

4.3.1 Type of Mutations Tolerated

A substring u is considered similar to the substring ρ^p if it can be written as $u = u_1u_2\cdots u_p$ where each word u_k $(k = 1\cdots p)$ is obtained by at most ε mutations on ρ and where ε is some pre-specified limit in the range $0 \leq r \leq |\rho|$. This was explained in the previous paragraph (Section 4.3). (Note that in running FireμSat, the user has further options for constraining the search for ATRs. These options are discussed in Section 4.3.2. They are concerned with constraining the ratio of ATREs to PTREs in a string and/or constraining the number of consecutive ATREs in the string.)

To further illustrate the above, consider an example based on the three letter PTRE ρ = ACG, where $\varepsilon = 1$ has been selected. This means that at most 1 mutation is allowed. The authorized forms of each ATRE, u_k, are therefore as follows:

1. The word ρ itself: u_k = ACG and $|u_k| = 3$.
2. The word ρ with the mismatch of one base: $u_k \in \{\texttt{XCG}|\texttt{X} : \{\texttt{C,G,T}\}\} \cup \{\texttt{AXG}|\texttt{X} : \{\texttt{A,G,T}\}\} \cup \{\texttt{ACX}|\texttt{X} : \{\texttt{A,C,T}\}\}$. In all these cases $|u_k| = 3$.
3. The word ρ with the deletion of one base: $u_k \in \{\texttt{CG}, \texttt{AG}, \texttt{AC}\}$. Thus, in all these cases $|u_k| = 2$.
4. The word ρ with an insertion in front of any position ρ_i of ρ. $u_k \in \{\texttt{ACGX}|\texttt{X} : \{\texttt{A,C,G,T}\}\} \cup \{\texttt{ACXG}|\texttt{X} : \{\texttt{A,C,G,T}\}\} \cup \{\texttt{AXCG}|\texttt{X} : \{\texttt{A,C,G,T}\}\} \cup \{\texttt{XACG}|\texttt{X} : \{\texttt{A,C,G,T}\}\}$. In all these cases $|u_k| = 4$.

This manner of defining authorized forms of mismatches and deletions of u_k derives from experimental observations cited by Rivals *et al.* (1995). It has been endorsed by Benson (1999) as providing statistically relevant information.

It should be noted that all these words keep at least 2 bases from the original word ρ.

The definition of insertions given above does not fully correspond to that of Rivals *et al.* (1995). The latter would consider any element of the following set to be insertions for the motif ACG:

$\{\texttt{ACGX}|\texttt{X} : \{\texttt{A,C,G,T}\}\} \cup \{\texttt{ACX}|\texttt{X} : \{\texttt{A,C,G,T}\}\} \cup \{\texttt{AGX}|\texttt{X} : \{\texttt{A,C,G,T}\}\} \cup \{\texttt{CGX}|\texttt{X} : \{\texttt{A,C,G,T}\}\} \cup \{\texttt{XCGY}|\texttt{X} : \{\texttt{C,G,T}\}; \texttt{Y} : \{\texttt{A,C,G,T}\}\} \cup \{\texttt{AXGY}|\texttt{X} : \{\texttt{A,G,T}\}; \texttt{Y} : \{\texttt{A,C,G,T}\}\} \cup \{\texttt{ACXY}|\texttt{X} : \{\texttt{A,C,T}\}; \texttt{Y} : \{\texttt{A,C,G,T}\}\}$.

Thus, Rivals *et al.* (1995) consider an insertion to be any word derived by concatenating a base to the end of a PTRE, or to the end of an ATRE that is either a deletion or a mismatch in the sense defined above.

It should be noted that the decision to part company with Rivals *et al.* (1995) on the matter, of how to handle insertions was not taken lightly. Indeed, the approach was discussed with two molecular biologists who were positive about the statistical relevance of the information that would be generated by the proposed algorithm.

In all the above motif error cases $3 \leq |u_k| \leq 4$. It is clear that in some cases (e.g. a character concatenated after a mismatch has occurred), that if the motif length is three, then Rivals *et al.* (1995) allows for a motif error of two. However, our convention (to be further discussed in Section 4.3.2) only allows one motif error per motif when the motif length is 3.

As it stands, the foregoing could lead to ambiguity in determining the mutational origin of a string. For example, `ACG` could be construed as some intended PTRE, ρ, or as a deletion of the last nucleotide, `G`, of the PTRE ρ, followed by the insertion of `G`.

To resolve such ambiguities, the following rules will be applied wherever possible:

1. A string will be interpreted as a PTRE rather than as an ATRE with mutations.

2. A string will always be regarded as an ATRE that results from mismatches, rather than from insertions or deletions.

3. An ATRE will be regarded as originating from a deletion rather than from an insertion.

The algorithm proposed also allows for other types of errors that can be adjusted by the user. More details pertaining to this matter are to be found in Section 4.3.2.

In principle then, an algorithm seeking TRs could rely on the motif error (ε) alone to determine, when the end of a candidate string has been found. However, in practice it is useful to rely on additional metrics. Section 4.3.2 introduces three such metrics. The section indicates how to determine whether a string that has been found to be a possible TR at some point in the algorithm, should be output as such, or whether further processing should occur to see if the string can be further extended to produce a longer TR.

4.3.2 Additional Metrics and Threshold Values

In addition to considering ε (the maximum motif error that may occur within a TR), FireμSat also computes three additional metrics. These are σ, the so-called

substring error, α, the maximum number of ATREs that may occur consecutively, and β, the minimum number of TREs to flag reporting.

In each case, the user can specify maximum values for these thresholds, which FireμSat will use as a threshold value in determining when a given substring can be regarded as a TR. The details of these thresholds will now be provided.

1. *The substring error*: σ
   This is a measure that provides additional useability and flexibility to the user. The FireμSat software provides the user with the option to either calculate the relative substring error or the absolute substring error. These measures are computed at appropriate points by FireμSat and then compared against a user-specified threshold value of the *maximum substring error allowed*, τ. During processing $\sigma \leq \tau$ should always hold.

   In line with the guidelines suggested by Benson (1999), the value of σ depends, *inter alia* on penalties (or weights) allocated by the user to mismatches (p_m), deletions (p_d) and insertions (p_i). A definition and a description of the two substring errors follows:

   - *The absolute substring error*: σ
     For a given motif, ρ, and a given substring that has been partitioned into the form $u = \rho u_2 \cdots u_p$, σ on u is computed as:

     $$\sigma = (n_d * p_d) + (n_i * p_i) + (n_m * p_m)$$

     where n_d is the number of deletions in u; n_i is the number of insertions in u and n_m is the number of mismatches in u. σ is calculated after each detection of a TRE. After σ has been calculated it is compared against a user specified threshold value, τ. During execution for each TR,s if $\sigma \leq \tau$ does not hold, then execution is terminated and the detected TR is reported on, provided that the other conditions discussed hereafter, hold. The user may rely on system default values for the penalties. These are $p_i = 1.0$, $p_d = 1.0$ and $p_m = 0.5$ respectively. A penalty weight of 0 may be chosen for one or more of the mutation types, in which case no penalty is assigned to ATREs that derive from that mutation type.

     The absolute substring error enables the user to assign values to penalties in such a manner that it is possible for FireμSat$_2$ to output for example only TRs that consist of PTREs and ATREs with mismatches. If sufficient TRs are not detected, then the deletion penalty values and/or insertion penalty values can be adjusted, such that TRs containing mismatches, deletions and/or insertions will be detected and reported on.

- *The relative substring error*: σ
  This is a measure of the extent to which the number (weighted as described below) of ATREs in the candidate TR exceeds the number of PTREs. The measure is computed after each detection of a TRE by FireμSat$_2$ and then compared against a user-specified threshold value of the *maximum relative substring error allowed*, τ. During processing $\sigma \leq \tau$ should always hold.

  Similarly to the absolute substring error, the value of σ of the relative substring error also depends, *inter alia* on penalties (or weights) allocated by the user to mismatches (p_m), deletions (p_d) and insertions (p_i). For a given motif, ρ, and a given substring that has been partitioned into the form $u = \rho u_2 \cdots u_p$, σ on u is computed as:

$$\sigma = (n_d * p_d) + (n_i * p_i) + (n_m * p_m) - n_{ptre}$$

  where n_d is the number of deletions in u; n_i is the number of insertions in u; n_m is the number of mismatches in u; and n_{ptre} is the number of PTREs in u.

  The user may rely on the same system default values as in the case of the absolute substring error.

  In the case of the relative substring error the value of σ therefore reflects the extent to which the number of ATREs exceeds the number of PTREs, weighted in terms of penalty values associated with mismatches, deletions and insertions. A restrictive default value has not been allocated to τ. If the user does not enter a value for τ then the substring error, σ, may be considered to be able to evaluate to infinity. Note, the relative substring error is similar to the original substring error defined for FireμSat, in De Ridder *et al.* (2006a) and De Ridder *et al.* (2006b)

The foregoing implies that FireμSat has to keep a count of the number of the various types of mutations. As will be seen in Section 4.4.2, FireμSat$_1$ makes use of an FA denoted by FA_ρ, which is the sum (in the sense of Kleene's theorem discussed in Section 4.2) of four other FAs: one for recognizing PTREs, and one each for recognizing insertions, deletions and mismatches. In general the substring error σ is calculated every time a final state is reached in FA_ρ. Each such final state is associated with a unit increment in either the number of PTREs, or the number of insertions, or the number of deletions or the number of mismatches. It is these final states, therefore, that enable the counting of the various types of mutations.

For as long as $\sigma \leq \tau$ holds, the scan of the input string continues in an effort to increase the length of the TR found to date. If the condition is

not met, then the TR found to date is output and the next TR in the input string is sought.

Of course, whenever a sink state (see Section 4.2) of FA_ρ is reached, then the TR is also output and the search for the next TR resumed.

2. *The maximum number of consecutive ATREs*: α
The user has the option of entering a value denoted by α. This value indicates the maximum number of ATREs that may occur consecutively. Thus α serves as a second threshold value.

   If the user specifies a value for α, then the counter tn_{atreC} is maintained to record the total number of consecutive ATREs since the last PTRE.

   The counter is incremented whenever an ATRE has been read (indicated by a transition to a final state of FA_ρ) irrespective of the type of elements — whether it be an insertion, deletion or mismatch. However, when a PTRE is read, then the value of tn_{atreC} is again set to zero. The processing of a string will only proceed if $tn_{atreC} \leq \alpha$.

   Note that a value for α is not activated by default. Thus, if the user does not enter a value for α, then there is no limit to the number of ATREs that may occur consecutively. (Alternatively, one might say that the default value of α is ∞.)

3. *The minimum number of tandem repeat elements*: β
To avoid the output of unwanted data, the user may indicate the minimum number of TREs that has to occur before a TR is output, denoted by β. To this end, a count tn_{tre}, is kept of the total number of TREs encountered to date in the current candidate TR. In fact, a count is also kept of the total number of PTREs encountered to date, tn_{ptre} and of the total number of ATREs encountered to date, tn_{atre}. Clearly $tn_{tre} = tn_{ptre} + tn_{atre}$.

   The current candidate TR will only be reported as a TR if one of the previously mentioned thresholds or terminating conditions is encountered *and* if $tn_{tre} \geq \beta$. The default value for β is two.

To illustrate these concepts, consider the genetic substring:
`ACGCGCGCGACGACTACGACT`.
Let the motif be `ACG`. The values for n_d, n_i, n_m, tn_{ptre}, tn_{atre} and tn_{atreC} are as follows at different processing intervals of the substring.

0.	ACGCGCGCGACGACTACGACT	n_d	n_i	n_m	tn_{ptre}	tn_{atre}	tn_{atreC}
1.	ACG	0	0	0	1	0	0
2.	ACGCG	1	0	0	1	1	1
3.	ACGCGCG	2	0	0	1	2	2
4.	ACGCGCGCG	3	0	0	1	3	3
5.	ACGCGCGCGACG	3	0	0	2	3	0
6.	ACGACACACACGCGC	3	0	1	2	4	1
7.	ACGCGCGCGACGACTACG	3	0	1	3	4	0
8.	ACGCGCGCGACGACTACGACT	3	0	2	3	5	3

Suppose that τ was specified by the user as 5; ε and that the default values for the penalties are used, namely $p_i = 1.0, p_d = 1.0, p_m = 0.5$. Then:

1. the absolute substring error will be calculated as follows:

$$\begin{aligned} \sigma &= (n_d * p_d) + (n_i * p_i) + (n_m * p_m) \\ &= (3 * 1) + (0 * 1) + (2 * 0.5) \\ &= 4 \end{aligned}$$

2. the relative substring error will be calculated as follows:

$$\begin{aligned} \sigma &= (n_d * p_d) + (n_i * p_i) + (n_m * p_m) - n_{ptre} \\ &= (3 * 1) + (0 * 1) + (2 * 0.5) - 3 \\ &= 1 \end{aligned}$$

and since this is less than the specified value for τ, FireμSat would attempt to explore elements beyond the given genetic substring before deciding at which stage the substring should be reported as a TR.

4.4 Algorithm Construction

Section 4.4 introduces the three different FireμSat algorithms by providing algorithmic details in Dijkstra's guarded command language (GCL). Section 4.4.1 provides GCL background knowledge and includes pointers to the lay out of the remainder of Section 4.4.

4.4.1 A Brief Introduction to FireμSat

The theory underlying FireμSat is a combination of straightforward FA technology, combined with a flavour of Moore machine technology. Three different

algorithms that detect TRs in DNA are presented in Dijkstra's guarded command language (GCL).

In the presented GCL the semantics of the if-statement specifies that non-deterministic selection of the guards takes place if more than one guard evaluates to true. Therefore to avoid ambiguity, guards have to be designed to be mutually exclusive. Guidelines for the development of GCL have been obtained from [Kourie (2009); Vide *et al.* (2003) and Dahl *et al.* (1972)].

FireμSat$_1$ is explained in Section, 4.4.2. A discussion of FireμSat$_2$ follows in Section, 4.4.3. Finally FireμSat$_3$ is introduced in Section, 4.4.4. The number of traversals of the input string of both FireμSat$_1$ and FireμSat$_2$ depend on the number of motifs or PTREs for which the genetic input string should be scanned. There is exactly one traversal for each motif. Several different TRs may be identified within the same substring of the input string. Consider for example the TR sub string:
ACGACGACGACGACGACGACG
This TR will be reported on three times with three different introductory PTREs namely *ACG*, *CGA* and *GAC* respectively. In contrast to FireμSat$_1$ and FireμSat$_2$, FireμSat$_3$ traverses the genetic input string only once for each relevant motif length, as indicated by the user. The user input for the three algorithms is similar and as follows:

- The lower and upper bound of motif lengths to be considered (l_{min} and l_{max} respectively);
- the maximum allowable motif error (ε — discussed in Section 4.3.1);
- the maximum allowable substring error (τ — discussed in Section 4.3.2);
- the penalty values used to calculate the substring error (p_m, p_d and p_i, all explained in Section 4.3.2);
- the maximum allowable number of ATREs that may occur consecutively (α — discussed in Section 4.3.2);
- the minimum number of TREs that should occur before a string is output as a TR (β — explained in Section 4.3.2); and
- the set (S) containing the genomic sequences (s) in FASTA format.

The algorithms return a set of tuples. Each tuple contains the following information about a detected TR in s:

- the start index of the the TR in s (pos);

- the length of the TR detected (len);
- the number of PTRE elements in the TR (n_{ptre}); and
- the number of mutations, deletions and insertions in the TR (n_m, n_d and n_i respectively).

Below the theoretical underpinning of FireμSat$_1$ is introduced. An indication is also given of how the theory has been applied.

4.4.2 Theory Underlying FireμSat$_1$

For illustrative purposes, the use of ACG as the motif string is continued. In addition, to facilitate the explanation of the algorithm, the following FAs are introduced. In each case, a description is given of how the given FA scans a string of the form $u = \rho u_2 u_3 \cdots u_p$

- $FA_P(\rho)$ is an FA that reaches a final state after scanning the first occurrence of ρ in u. In fact it reaches the final state again if $u_2 = \rho$ is encountered in u, and again if $u_3 = \rho$ is encountered in u, etc. However, $FA_P(\rho)$ goes to a sink state (see Section 4.2) as soon as a character in u is encountered that indicates that u is not a PTR. Thus $FA_P(\rho)$ accepts a PTR of arbitrary length with motif ρ, entering the final states as many times as there are PTREs in the PTR. A diagrammatic representation of $FA_P(\texttt{ACG})$ is given in Figure 4.4.
- $FA_D(\rho, \varepsilon)$ is an FA that, upon scanning u, reaches its first final state once the substring ρ has been read. $FA_D(\rho, \varepsilon)$ continues to reach final states after scanning each word, u_i (where $i = 2 \cdots p$) provided that one of the following conditions hold: a) either $u_i = \rho$ or b) u_i is a word deduced from ρ that contains a maximum of ε deletions. However, $FA_D(\rho, \varepsilon)$ will move to a sink state as soon as it is apparent that neither of these two conditions hold. A diagrammatic representation of $FA_D(\texttt{ACG}, 1)$ is given in Figure 4.5.
- $FA_M(\rho, \varepsilon)$ is an FA that functions analogously to $FA_D(\rho, \varepsilon)$, except that it functions in terms of *mismatches* rather than deletions. A graphical representation of $FA_M(\texttt{ACG}, 1)$ is given in Figure 4.6.
- $FA_I(\rho, \varepsilon)$ is an FA that functions analogously to $FA_D(\rho, \varepsilon)$, except that it functions in terms of *insertions* rather than deletions.

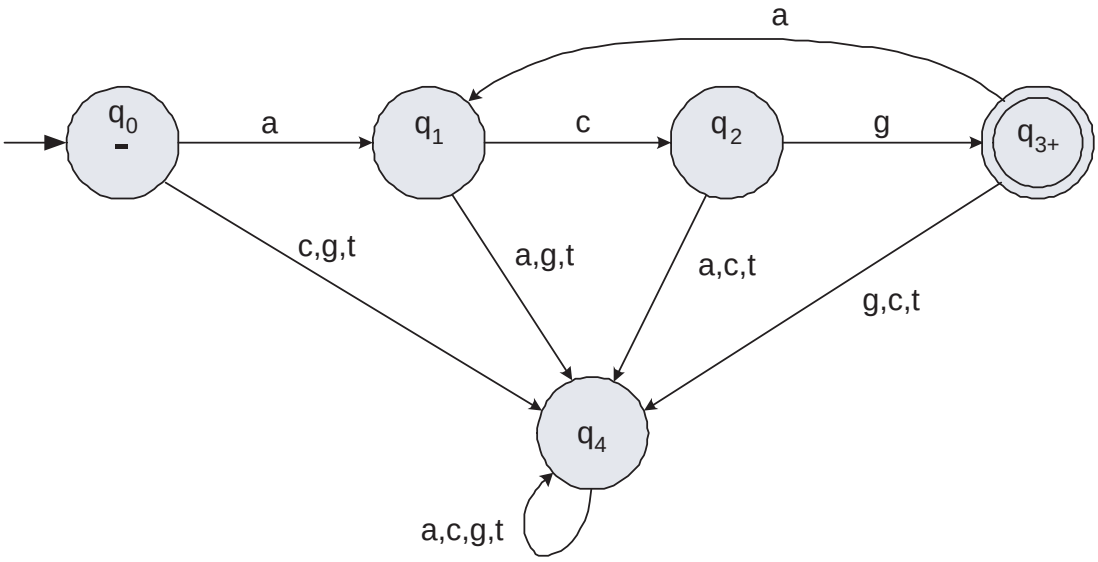


Figure 4.4: $FA_P(\texttt{ACG})$

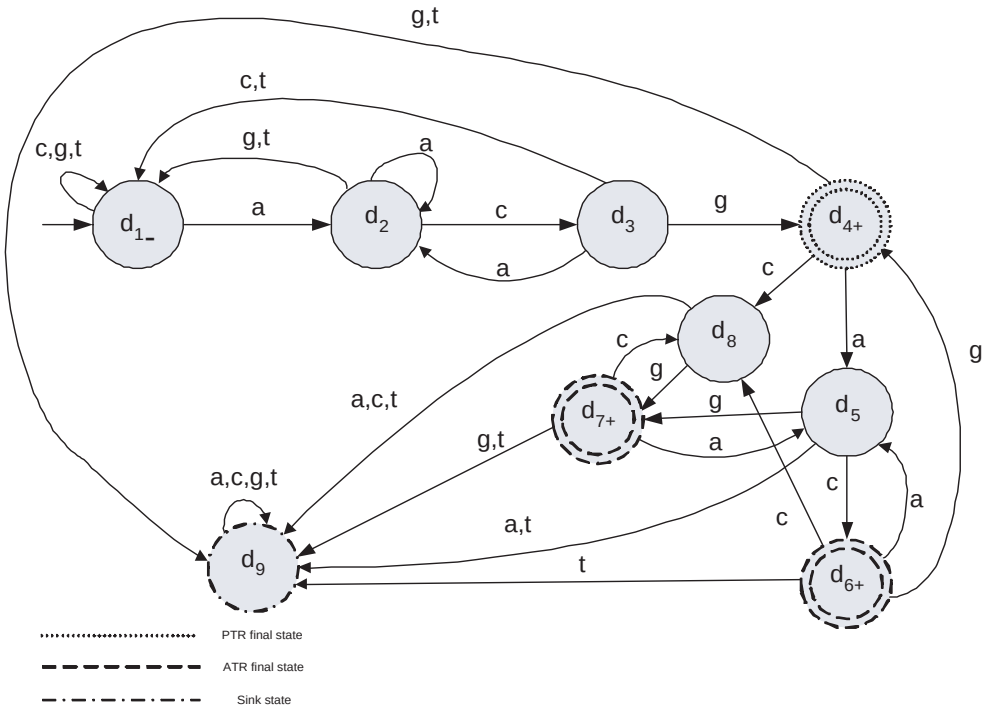


Figure 4.5: $FA_D(\texttt{ACG}, 1)$

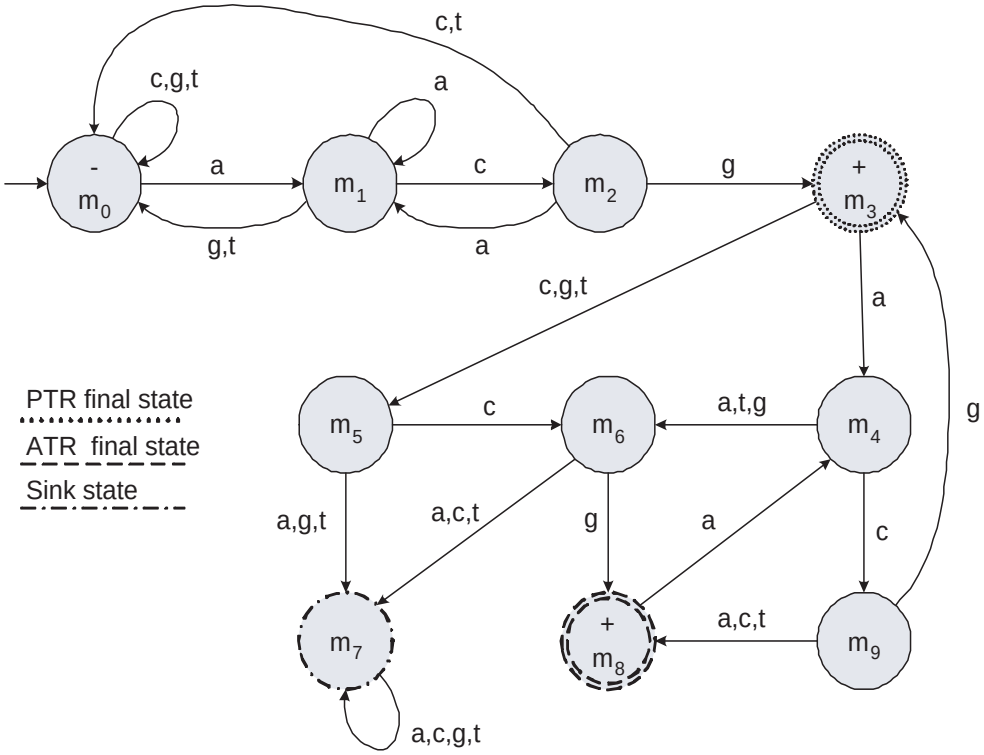


Figure 4.6: $FA_M(\texttt{ACG}, 1)$

- $FA_{\rho(l,\varepsilon)}$ is an FA obtained from the sum of all the previously defined FAs. Thus:

$$FA_{\rho(l,\varepsilon)} = FA_P(\rho) + FA_D(\rho,\varepsilon) + FA_M(\rho,\varepsilon) + FA_I(\rho,\varepsilon) \quad (4.1)$$

  where l is the motif length and ε indicates the number of motif errors allowed. In the interest of simplicity l as well as ε will be omitted in the remainder of the text. Thus for future reference l and ε are considered to be inherent to the intended motif, ρ. Equation 4.1 consequently reduces to:

$$FA_\rho = FA_P(\rho) + FA_D(\rho,\varepsilon) + FA_M(\rho,\varepsilon) + FA_I(\rho,\varepsilon) \quad (4.2)$$

For a given motif, it is relatively easy to specify the regular expressions that correspond to the various FAs just mentioned above. Although software packages are available to construct FAs from given regexs and to add these FAs, (for example, Fire Station [Watson (1994)]) it was considered more convenient to implement purpose-built software for the present purposes.

For example, the language accepted by $FA_P(\texttt{ACG})$ can be defined by the regex $(\texttt{ACG})(\texttt{ACG})^*$. Similarly, the languages accepted by $FA_D(\texttt{ACG},1)$, $FA_M(\texttt{ACG},1)$ and $FA_I(\texttt{ACG},1)$ may be defined by means of regexs, respectively, as follows:

- $FA_D(\texttt{ACG},1)$ accepts the language defined by the regex $(\texttt{ACG})(\texttt{ACG}+\texttt{AC}+\texttt{AG}+\texttt{CG})^*$.
- $FA_M(\texttt{ACG},1)$ accepts the language defined by the regex $(\texttt{ACG})(\texttt{ACG}+\texttt{CCG}+\texttt{GCG}+\texttt{TCG}+\texttt{AAG}+\texttt{AGG}+\texttt{ATG}+\texttt{ACA}+\texttt{ACC}+\texttt{ACT})^*$.
- $FA_I(\texttt{ACG},1)$ accepts the language defined by the regex $(\texttt{ACG})(\texttt{ACG}+\texttt{AACG}+\texttt{CACG}+\texttt{GACG}+\texttt{TACG}+\texttt{ACCG}+\texttt{AGCG}+\texttt{ATCG}+\texttt{ACAG}+\texttt{ACGG}+\texttt{ACTG}+\texttt{ACGA}+\texttt{ACGC}+\texttt{ACGT})^*$.

In general, then, an FA can be set up along the lines indicated above for an arbitrary motif, ρ. A trace through such an FA will confirm that strings of the form $p\rho u_1 \cdots u_q$ are recognized, where:

- p is some arbitrary non-motif prefix preceding a TR,
- ρ is the motif (in the present example, ACG) of the TR,

- each $u_i, i = 1 \cdots q$ is an ATRE based on ρ, allowing for an error of maximally ε. Note that in the present example, $\rho = \texttt{ACG}$ and thus $|\rho| = 3$. Therefore, as previously discussed, ε is only allowed to assume the value of 1 or 0. Thus only 1 or 0 deletion or mismatch may occur in each respective TRE of each figure, and

- $q \geq 1$ is the number of ATREs that follow on from the motif or PTRE in the TR.

It will also be seen that if any additional element that does not belong to the TR identified up to that point is encountered in the input string, then the FA transits to a sink state in each respective case. State q_4 in Figure 4.4, state d_9 in Figure 4.5 and state m_7 in Figure 4.6 are sink states.

In Figures 4.5 and 4.6, there are also two kinds of final states: those which signal that a motif (PTRE) has been scanned and those which indicate that a deletion (or mismatch) has been scanned. These states will be referred to as PTR- and ATR-final states, respectively. As explained below, the number of transitions into these states have to be counted, and the respective values of σ, tn_{atreC} and tn_{tre} have to be correspondingly updated, so as to ensure that the strings designated as TRs are consistent with thresholds τ, α and β respectively, as explained in Section 4.3.2 above.

In order to construct $FA_{\rho(|\rho|,1)}$ we first construct the respective constituent machines and then apply the constructive algorithm which forms part of the proof of *Rule 2, Part 3* of Kleene's theorem discussed in Section 4.2.
$FA_{\texttt{ACG}} = FA_P(\texttt{ACG}) + FA_D(\texttt{ACG}, \texttt{1}) + FA_M(\texttt{ACG}, \texttt{1}) + FA_I(\texttt{ACG}, \texttt{1})$.

This can be done by using the Fire Engine software [Watson (1994)] that provides a toolkit for generating and adding FAs. FA toolkits in general are discussed in more detail in Watson (1995), Watson (1996), Watson (2001) and Watson (2002).

The discussion to date can be generalized: $FA_{\texttt{XYZ}}$ is an FA for recognizing the parameterized motif XYZ of length 3. The parameters are X, Y and Z and each of these parameters can be instantiated to any one of the nucleotides $\{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$. This parameterized FA is, as before, the sum of four other FAs, each of which are also parameterized.

Thus the regex associated with $FA_P(\texttt{XYZ})$ can be defined as $(\texttt{XYZ})(\texttt{XYZ})^*$. Furthermore $FA_D(\texttt{XYZ}, \texttt{1})$, $FA_M(\texttt{XYZ}, \texttt{1})$ and $FA_I(\texttt{XYZ}, \texttt{1})$ can also be defined as follows:

- $FA_D(\texttt{XYZ}, \texttt{1})$ accepts the language defined by the regex $(\texttt{XYZ})(\texttt{XYZ} + \texttt{XY} + \texttt{XZ} + \texttt{YZ})^*$.

- $FA_M(\texttt{XYZ}, \texttt{1})$ accepts the language defined by the regex $(\texttt{XYZ})(\texttt{XYZ} + \texttt{YYZ} + \texttt{ZYZ} + \texttt{RYZ} + \texttt{XXZ} + \texttt{XZZ} + \texttt{XRZ} + \texttt{XYX} + \texttt{XYY} + \texttt{XYR})^*$.

- $FA_I(\texttt{XYZ}, 1)$ accepts the language defined by the regex $(\texttt{XYZ})(\texttt{XYZ} + \texttt{XXYZ} + \texttt{YXYZ} + \texttt{ZXYZ} + \texttt{RXYZ} + \texttt{XYYZ} + \texttt{XZYZ} + \texttt{XRYZ} + \texttt{XYXZ} + \texttt{XYZZ} + \texttt{XYRZ} + \texttt{XYZX} + \texttt{XYZY} + \texttt{XYZR})^*$.

Thus, in principle, any FA_ρ of motif length 3 can be algorithmically constructed. Similarly parameterized versions for FA_ρ can be constructed for $|\rho| = 2, 3, 4, 5$ and for permissible values of ε. In each case the regexs relating to the constituent FAs have to be determined, the corresponding FAs are then derived using tools, programmed by Mr. A.P.F. Marais (an honours project student of the author), similar to the Fire Engine toolkit and these derived FAs are then summed, also using the toolkit, to provide FA_ρ. Algorithm 4.4.1 illustrates how the respective regular expressions, ζ, needed to construct the FA_ρ's are generated. FA_ρ's are required during the run of FireμSat$_1$.

An explanation of some of the functions of Algorithm 4.4.1 is as follows:

- the function $rep(s, p, c)$ replaces the character at position p, in string s, with character c and returns the resultant string;
- the function $del(s, p)$ deletes the character at position p, in string s and returns the resultant string; and finally
- the function $ins(s, p, c)$ inserts the character c at position p, into string s and returns the resultant string.

After the completion of each of the regex construction loops, it is necessary to delete the last added + sign and to close the opening brackets of the current type. From the code it is apparent that the + sign is added to make provision for expansion after each element.

Note, the example function, $genRegEx(\rho, \Sigma) : string$ returns a regex in the form of a string as indicated. The author is aware thereof that this algorithm can also be implemented recursively.

Algorithm 4.4.1
$\Sigma :=$ "$acgt$"
func $genRegEx(\rho, \Sigma) : string$
var ζ, i, j

$i = |\rho|$
$; \zeta :=$ "(" $+ \rho +$ ")("

$\{Mismatches\}$
$; i = |\rho|$

;**do** $(i > 0) \rightarrow$
  $j = |\Sigma|$
  ;**do** $(j > 0) \rightarrow$
    if $(\rho[i] \neq \Sigma[j]) \rightarrow$
      $\zeta := \zeta + rep(\rho, i, \Sigma(j)) + " + "$
    [] $(\rho[i] = \Sigma[j]) \rightarrow$ skip
    fi
    ; $j := j - 1$
  od
  ; $i := i - 1$
od
; $\zeta := del(\zeta, |\zeta|) + ") + ("$

$\{Deletions\}$
; $i = |\rho|$
;**do** $(i > 0) \rightarrow$
  $\zeta := \zeta + del(\rho, i)$
  ; $i := i - 1$
od
; $\zeta := del(\zeta, |\zeta|) + ") + ("$

$\{Insertions\}$
; $i = |\rho|$
;**do** $(i > 0) \rightarrow$
  $j = |\Sigma|$
  do $(j > 0) \rightarrow$
    $\zeta := \zeta + ins(\rho, i, \Sigma(j)) + " + "$
    ; $j := j - 1$
  od
  ; $i := i - 1$
od
; $\zeta := del(\zeta, length(\zeta)) + ")"$
; **return** ζ
cnuf

Once FA_ρ has been constructed, certain adaptations to the conventional FA language recognition algorithm are required, when scanning through a genomic sequence in search of the next TR. Some of the details relating to these adaptations will be discussed later. For the present, consider the high-level description of FireμSat$_1$ given in Algorithm 4.4.2. The input as well as the output parameters were discussed in Section 4.4.1.

The following functions are assumed:

- $genMotifs(l)$ generates a set of all words from the alphabet $\Sigma = \{\texttt{A,C,G,T}\}$ of the length (l) specified by the user.
- $genFA(l, \varepsilon)$, $genFA(genRegEx(l, \varepsilon))$ returns FA, where l indicates the motif length and ε indicates the number of motif errors allowed. Note, in the case of FireμSat$_1$ $genRegEx$, Algorithm 4.4.1, is used to obtain a regex before generating the required FA. In the case of FireμSat$_2$ and FireμSat$_3$ the machines are directly generated in line with the user specified ρ and ε.
- $decorate(FA, \rho)$ decorates the generated FA with the motif ρ.
- $find(s, pos, \rho)$ returns the index of the start of the next candidate, TR of motif ρ, in the suffix of sequence s that starts at pos. It returns -1 if no match is found.
- $computeTR1(s, pos, FA_\rho, \tau, \alpha, \beta, p_m, p_d, p_i)$. $computeTR1$ returns a tuple of information about the next TR that is recognised by FA_ρ within the constraints specified by τ, α and β as explained in Section 4.3.2. The tuple contains the start position of the TR (pos), its length ($trlen$) and the count of the numbers of PTREs, mismatches, deletions and insertions (n_{ptre}, n_m, n_d and n_i). A detailed discussion of $computeTR1$ will follow before its GCL-code is introduced. $computeTR1$ calls the function $isEnd$.
- $isEnd(Q, \tau, \alpha, \beta, p_m, p_d, p_i, n_m, n_d, n_i, tn_{ptre}, tn_{atre}, tn_{atreC}, back)$. The purpose of the function $isEnd$ is threefold:
  - $isEnd$ determines if $computeTR1$ should continue to process the next character of a detected TR.
  - The function $isEnd$ determines if a detected TR should be reported on or not.
  - Finally, $isEnd$ indicates if the last detected TRE should be added to the currently detected TR or not.

  A detailed discussion of $isEnd$ will precede its GCL-code in Section 4.4.2.2.

Algorithm 4.4.2
func $Fire\mu Sat_1(l_{min}, l_{max}, \varepsilon, \tau, \alpha, \beta, p_m, p_d, p_i, S) : tuples$
Pre : $\{(0 < l_{min} \leq l_{max}) \wedge (0 \leq \varepsilon) \wedge (\sigma \leq \tau) \wedge (0 \leq tn_{atreC} \leq \alpha)$
$\wedge (\forall s \in S : length(s) > l) \wedge (0 \leq \beta \leq tn_{tre}) \wedge (S \in \Sigma^*)\}$
Post : $\{(pos, len) \in tuples \equiv \exists \rho : \Sigma^* \cdot |\rho| \in [l_{min}, l_{max}]$
$\wedge \forall s \in S : (\forall i \in [pos, pos + len) : \neg isEnd(s[i], \varepsilon, \tau, \alpha, \beta, p_m, p_d, p_i, \rho))$

$\wedge \forall s \in S : (\neg isEnd(s[pos + len], \varepsilon, \tau, \alpha, \beta, p_m, p_d, p_i, \rho))\}$
var i, ρ, pos, len, s

$tuples := \emptyset$
; **for** $l \in [l_{min}, l_{max}] \rightarrow$
  $FA := genFA(genRegEx(l, \varepsilon))$
  $; W := genMotifs(l)$
  for $s \in S \rightarrow$
    for $\rho \in W \rightarrow$
      $FA_\rho := decorate(FA, \rho)$
      $; pos := find(s, 0, \rho)$
      ; **do** $(pos \neq -1) \rightarrow$
        $< pos, len, n_{ptre}, n_m, n_d, n_i > :=$
          $computeTR1(s, pos, FA_\rho, \tau, \alpha, \beta, p_m, p_d, p_i)$
        ; **if** $(len > |\rho|) \rightarrow$
          $tuples := tuples \ \cup \{< pos, len, n_{ptre}, n_m, n_d, n_i >\}$
        [] $(len \leq |\rho|) \rightarrow$
          $len := 0$
        fi
        ; $pos := find(s, pos + len + 1, \rho)$
      od
    rof
  rof
rof
; **return** *tuples*
cnuf

4.4.2.1 Explanation of $computeTR1$

$computeTR1$ updates various counters, initially at 0, after a motif has been encountered as we scan through a string. The GCL of $computeTR1$ is presented in Algorithm 4.4.3.

Note that in order to use FA_ρ appropriately in FireμSat$_1$, it is required that the final states of the original component FAs be identifiable in it. It was pointed out previously that one of the features of the constructive algorithm introduced in the proof of *Rule 2, Part 3* of Kleene's algorithm (Section 4.2) is, that if it is used to compute say $FA_X = FA_Y + FA_Z$, then every final state in FA_Y can be mapped to a final state in FA_X. The same holds true for every final state in FA_Z. Moreover, every final state in FA_X will either map to a final state in FA_Y, or to a final state in FA_Z, or to a final state in both FA_Y and FA_Z.

To determine whether the conditions on the threshold values, τ (representing the maximum allowable substring error), α (representing the maximum number of ATREs that may occur consecutively) and β (the minimum allowable number of TREs that have to occur before a TR is reported) have been met, when scanning through a TR, various counters initially at 0, have to be updated once a motif is encountered as we scan through a string. To this end, let the variables tn_{ptre} and tn_{atre} store the number of PTR-final states and ATR-final states encountered to date, respectively. Additionally, the variables n_d, n_m and n_i store the number of deletions, mismatches and insertions encountered to date, respectively.

The logic of how these counters are to be updated whenever a state, R, of an FA_ρ is applied in $computeTR1$ as given in Algorithm 4.4.3 below.

Note specifically, that more than one of these conditions may hold for a final state, as discussed in Section 4.2.

It will be seen that if a final state is of multiple types, then the PTRE counter (tn_{ptre}) takes precedence, followed by the mismatch counter (n_m), followed by the deletions counter (n_d), followed by the insertion counter (n_i). By this is meant that if a state is encountered that is final for both PTREs and mismatches, then the PTRE counter is incremented rather than the mismatch counter. Similarly, mismatches are incremented rather than deletions, etc. More details regarding the practical implementation of the precedence rules follows after the GCL-code of a simplified version of $computeTR1$.

Note in passing that the semantics of GCL dictates that if a condition arises that does not fire a guard in an if-statement, then the if-statement will abort, indicating that such a condition constitutes an error. Thus, for example in the code below, there is no guard to deal with a condition where a state is designated as final, but it is not associated with a PTRE, nor with a mismatch, nor with a deletion, nor with an insertion. Such a condition ought not to arise, and would indeed constitute an error if it did.

$computeTR1$ as presented below, in somewhat simplified form, is called by both $\mathsf{Fire}\mu\mathsf{Sat}_1$ and $\mathsf{Fire}\mu\mathsf{Sat}_3$. The purpose of $computeTR1$ is to detect a TR and return its length, number of PTREs and number of mismatches, deletions and insertions. In this context the processing of a TR includes the incrementing of the applicable counters. The procedure $computeTR1$ will continue to execute until the end of a TR is reported by $isEnd$ (presented in Algorithm 4.4.3) or until the end of the input set S is reached. The input/output parameter *back* is passed to $isEnd$. Note that *back* is the offset by which the length of the TR found to date should be decreased if $isEnd$ reports that the end of the TR has been reached.

Algorithm 4.4.3
func $computeTR1(s, pos, FA_\rho, \tau, \alpha, \beta, p_m, p_d, p_i) : tuple$

$pos, R := pos + 1, nextState(FA_\rho, R, s[pos])$
$; n_m, n_d, n_i, tn_{ptre}, tn_{atre}, tn_{atreC}, back := 0, 0, 0, 0, 1, 0, 0$
; **do** $(\neg isEnd(R, \tau, \alpha, \beta, p_m, p_d, p_i, n_m, n_d, n_i, tn_{ptre}, \beta, \alpha, back) \wedge \neg s(EOF)) \rightarrow$
if $(R \in PTRE) \rightarrow$
$tn_{ptre}, tn_{atreC} := tn_{ptre} + 1, 0$
$[\!]$ $(R \notin PTRE \wedge R \in Mis) \rightarrow$
$tn_{atre}, tn_{atreC}, n_m := tn_atre + 1, tn_{atreC} + 1, n_m + 1$
$[\!]$ $(R \notin (PTRE \cup Mis) \wedge R \in Del) \rightarrow$
$tn_{atre}, tn_{atreC}, n_d, := tn_{atre} + 1, tn_{atreC} + 1, n_d + 1$
$[\!]$ $(R \notin (PTRE \cup Mis \cup Del) \wedge R \in Ins) \rightarrow$
$tn_{atre}, tn_{atreC}, n_i, := tn_{atre} + 1, tn_{atreC} + 1, n_i + 1$
$[\!]$ $(R \notin (PTRE \cup Mis \cup Del \cup Ins) \wedge R \in MisMis) \rightarrow$
$tn_{atre}, tn_{atreC}, n_m, := tn_{atre} + 1, tn_{atreC} + 1, n_m + 2$
$[\!]$ $(R \notin (PTRE \cup Mis \cup Del \cup Ins \cup MisMis) \wedge R \in MisDel) \rightarrow$
$tn_{atre}, tn_{atreC}, n_d, n_m := tn_{atre} + 1, tn_{atreC} + 1, n_d + 1n_m + 1$
$[\!]$ $(R \notin (PTRE \cup Mis \cup Del \cup Ins \cup MisMis \cup MisDel) \wedge R \in MisIns) \rightarrow$
$tn_{atre}, tn_{atreC}, n_i, n_m := tn_{atre} + 1, tn_{atreC} + 1, n_i + 1n_m + 1$
$[\!]$ $(R \notin (PTRE \cup Mis \cup Del \cup Ins \cup MisMis \cup MisDel \cup MisIns)$
$\wedge R \in DelDel) \rightarrow$
$tn_{atre}, tn_{atreC}, n_d := tn_{atre} + 1, tn_{atreC} + 1, n_d + 2$
$[\!]$ $(R \notin (PTRE \cup Mis \cup Del \cup Ins \cup MisMis \cup MisDel \cup MisIns \cup DelDel)$
$\wedge R \in DelIns) \rightarrow$
$tn_{atre}, tn_{atreC}, n_d, n_i := tn_{atre} + 1, tn_{atreC} + 1, n_d + 1, n_i + 1$
$[\!]$ $(R \notin (PTRE \cup Mis \cup Del \cup Ins \cup MisMis \cup MisDel \cup MisIns \cup DelDel$
$\cup DelIns) \wedge R \in InsIns) \rightarrow$
$tn_{atre}, tn_{atreC}, n_i, n_m := tn_{atre} + 1, tn_{atreC} + 1, n_i + 1, n_m + 1$
$[\!]$ $(R \notin (PTRE \cup Mis \cup Del \cup Ins \cup MisMis \cup MisDel \cup MisIns \cup DelDel$
$\cup DelIns \cup InsIns) \rightarrow skip$
fi
$; pos, R := pos + 1, nextState(FA_\rho, R, s[pos])$
od
$; len := pos - pos - back$
; **return** $< pos, len, n_{ptre}, n_m, n_d, n_i >$
cnuf

Note that the function *isEnd*, called by *computeTR*1 is explained in detail in Section 4.4.2.2.

Furthermore, for the sake of simplicity the subsequent details regarding the analysis and incrementing of the final states have been omitted from Algorithm 4.4.3:

- The logic of the counter updates of the following state sets in Algorithm

4.4.3 are the same:

$R \in MisDel \equiv (R \in (MisDel \cup DelMis))$
$R \in InsDel \equiv (R \in (InsDel \cup DelIns))$
$R \in MisIns \equiv (R \in (MisIns \cup InsMis))$

Therefore the set names are equivalent to the respective unions as indicated above.

- Forward looking states in Algorithm 4.4.3 are required, if a longer length ATRE has a higher priority than the previously detected ATRE. For example, an ATRE containing a deletion will always be detected before the corresponding PTRE. The matter becomes more complex if the motif error $\varepsilon = 2$, is selected. If $\varepsilon = 2$ then $MisDel$, $DelMis$ and $DelDel$ should be implemented with forward looking states.

  To illustrate the three different instances where forward looking states are required:

  1. Consider the following guarded command in Algorithm 4.4.3:

     if $\cdots$
     $[\!]$ $R \notin (PTRE \cup Mis) \wedge R \in Del) \rightarrow$
     $tn_{atre}, tn_{atreC}, n_d, := tn_{atre} + 1, tn_{atreC} + 1, n_d + 1$
     $[\!]$ $\cdots$
     fi

     The deletion final state will be reached before a PTRE final state or a mismatch final state. Therefore, it is necessary to verify that the next character of the TRE under consideration is not a PTRE or an ATRE that can be considered to have a mismatch.
     Example: [Consider the PTRE ACG and the ATRE ACT a "copy" of "ACG". In FA_{ACG} the final state of the ATRE AC (an ATRE with a deletion) will be reached before the final state of the ATRE ACT (an ATRE containing a mismatch). Therefore, before the counter of the number of deletions is incremented, there should be "looked forward" to determine if the ATRE ACT is a mismatch or not. If the ATRE is a mismatch as in the case of ACT then the number of mismatches is incremented, else the number of deletions is incremented.]

     Similar examples could also be constructed for the remaining two cases.
     To make provision for the foregoing, the following logic is used:

     if $\cdots$
     $[\!]$ $R \notin (PTRE \cup Mis) \wedge R \in Del) \rightarrow$

$T := nextstate(FA_\rho, R, s[pos+1])$
if $(T \in (PTRE \cup Mis) \rightarrow$ skip
$[\!]$ $(T \notin (PTRE \cup Mis) \rightarrow$
$tn_{atre}, tn_{atreC}, n_d, := tn_{atre}+1, tn_{atreC}+1, n_d+1$
fi
$[\!]$ $\cdots$
fi

2. In a similar manner as explained in the previous item, forward looking states are also used to evaluate the equivalent states in $MisDel$ and $DelMis$.

   The implementation logic is as follows:

if $\cdots$
$[\!]$ $(R \notin (PTRE \cup Mis \cup Del \cup Ins \cup MisMis)$
$\wedge (R \in (MisDel \cup DelMis))) \rightarrow$
$T := nextstate(FA_\rho, R, s[pos+1])$
;**if** $(T \in (PTRE \cup Mis \cup MisMis) \rightarrow$ skip
$[\!]$ $(T \notin (PTRE \cup Mis \cup MisMis) \rightarrow$
$tn_{atre}, tn_{atreC}, n_d, n_m := tn_{atre}+1, tn_{atreC}+1, n_d+1, n_m+1$
fi
$[\!]$ $\cdots$
fi

3. Finally, forward looking into two future states is required in the the case of $DelDel$. The implementation logic is as follows:

if $\cdots$
$[\!]$ $(R \notin (PTRE \cup Mis \cup Del \cup Ins \cup MisMis \cup MisDel$
$\cup DelMis \cup MisIns \cup InsMis) \wedge (R \in DelDel)) \rightarrow$
$T := nextstate(FA_\rho, R, s[pos+1])$
$; U := nextstate(FA_\rho, T, s[pos+2])$
if $(U \in (PTRE \cup Mis \cup MisMis)) \rightarrow$ skip
$[\!]$ $(U \notin (PTRE \cup Mis \cup MisMis)) \rightarrow$
if $(T \in (DelMis \cup MisDel) \rightarrow$ skip
$[\!]$ $(T \notin (DelMis \cup MisDel) \rightarrow$
$tn_{atre}, tn_{atreC}, n_d := tn_{atre}+1, tn_{atreC}+1, n_d+2$
fi
$\cdots$
fi

In the case of FireμSat$_1$ *computeTR*1 traverses the genetic input string, update the relevant counters and calls the function *isEnd*, presented by Algorithm 4.4.4, which indicates whether the end of a TR is reached or not.

4.4.2.2 Explanation of *isEnd*

isEnd is called by *computeTR*1 as follows:
$isEnd(Q, \tau, \alpha, \beta, p_m, p_d, p_i, n_m, n_d, n_i, tn_{ptre}, tn_{atre}, tn_{atreC}, back)$.
Note that the input/output parameter *back* is the offset by which the length of the TR found to date should be decreased, if *isEnd* reports that the end of the TR has been reached. The purpose of the function *isEnd*, is threefold:

- *isEnd* determines if *computeTR*1 should continue to process the next character of a detected TR. If the next character of s should be processed as part of a current TR, then *isEnd* returns the value `false` to *computeTR*1, else *isEnd* returns the value `true` to *computeTR*1.
- The function *isEnd* determines if a detected TR should be reported or not. This is done by evaluating the conditions set by the different threshold values. If one or more of the conditions set by the threshold values are not met, then *isEnd* resets the counters accordingly and returns before returning `true` to *computeTR*1.
- Finally, *isEnd* indicates if the last detected TRE should be added to the currently detected TR or not. If a TR has been detected then *isEnd* calculates if the ratios of the different threshold values will still hold after the most recently detected TRE is added to the particular TR. If the ratios indicated by the user entered threshold values are not met, then the last TRE should be removed from the currently detected TR, and the value of *pos* in *computeTR*1 should be adjusted accordingly. The adjustment is accomplished by assigning the most recent TRE length to the output parameter *back*. It should be clear that *isEnd* will always return the value `true` to *computeTR*1 if a non-zero value has been allocated to *back*.

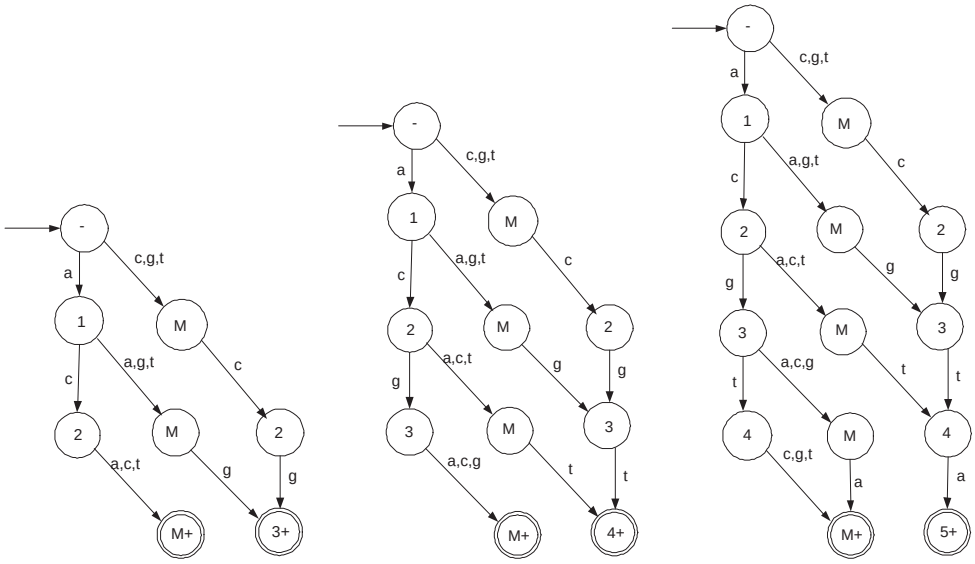
If *back* is assigned the value of the constant *len*, then that effectively indicates to *computeTR*1 that the current TR should be discarded, since the *computeTR*1 will compute the length of the most recently discovered TR to be $len - back$. This state of affairs is only attained when the end of a TR is reached, but the number of TREs is less than the prespecified minimum of β.

On the other hand, if *isEnd* is to report that the end of a TR has been reached, because one of the threshold values has been exceeded, then *back* is assigned the length of the last identified ATRE. *deadSet* is the set of sink states as defined in Section 4.2. The GCL of *isEnd* is provided in Algorithm 4.4.4.

Algorithm 4.4.4
func $isEnd(R, \tau, \alpha, \beta, p_m, p_d, p_i, n_m, n_d, n_i, tn_{ptre}, tn_{atre}, tn_{atreC}, |\rho|, back) : bool$
$vartrEnd := false$
$back := 0$
$;\sigma := (p_m \times n_m) + (p_d \times n_d) + (p_i \times n_i)$
if $(R \in deadSet) \rightarrow$
  $trEnd := true$
  $;$ **if** $(tn_{tre} < \beta) \rightarrow$
    $n_m, n_d, n_i, tn_{ptre}, tn_{atre}, tn_{atreC}, back := 0, 0, 0, 0, 0, len$
  $[\!]$ $(tn_{tre} \geq \beta) \rightarrow$ skip
  fi
$[\!]$ $((R \notin deadSet) \wedge (\sigma \leq \tau) \wedge (tn_{atreC} \leq \alpha)) \rightarrow$
  $trEnd := false$
$[\!]$ $((R \notin deadSet) \wedge ((\sigma > \tau) \vee (tn_{atreC} \leq \alpha))) \rightarrow$
  $trEnd := true$
  $;$ **if** $(tn_{tre} < \beta) \rightarrow$
    $n_m, n_d, n_i, tn_{ptre}, tn_{atre}, tn_{atreC}, back := 0, 0, 0, 0, 0, len$
  $[\!]$ $(tn_{tre} \geq \beta) \rightarrow$
    if $(R \in PTRE) \rightarrow$ skip
    $[\!]$ $(R \in Mis) \rightarrow$
      $n_m, tn_{atre}, tn_{atreC}, back :=$
        $n_m - 1, tn_{atre} - 1, tn_{atreC} - 1, |\rho|$
    $[\!]$ $(R \in Del) \rightarrow$
      $n_d, tn_{atre}, tn_{atreC}, back :=$
        $n_d - 1, tn_{atre} - 1, tn_{atreC} - 1, |\rho| - 1$
    $[\!]$ $(R \in Ins) \rightarrow$
      $n_i, tn_{atre}, tn_{atreC}, back :=$
        $n_i - 1, tn_{atre} - 1, tn_{atreC} - 1, |\rho| + 1$
    fi
  fi
fi
return $trEnd$
cnuf

Thus in summary, *isEnd* returns the boolean value `true` if one or more of the threshold values do not hold or if R is a sink state. Upon receiving the boolean value *true*, the loop in $computeTR1$ terminates. While R is not a sink state and all the threshold values hold, then the boolean value `false` is returned to $computeTR1$.

In the next section, Section 4.4.3, FireμSat$_2$ will be introduced. The key difference between FireμSat$_1$ and FireμSat$_2$ is that FireμSat$_1$ relies on various merged DFAs



(a) A DFA for length 3 (b) A DFA for length 4 (c) A DFA for length 5

Figure 4.7: Three DFAs that accept ATREs that contain one mismatch

and incorporates notions of counting states where as FireμSat$_2$ decorates several relatively small DFAs as will be seen in the next section.

4.4.3 Theory Underlying FireμSat$_2$

FireμSat$_2$ also relies on DFA technology. This theory will be elaborated on in the process of introducing the theoretical underpinnings of FireμSat$_2$. The DFAs used as running examples in this section cater for the detection of TRs that are introduced by any of the three motifs `ACG`, `ACGT` and `ACGTA`. In addition, to facilitate the explanation of FireμSat$_2$, the following DFAs are considered: $Mis(ACG)$, $Mis(ACGT)$ and $Mis(ACGTA)$. The graphical representation of these FAs can be found in Figures 4.7(a), 4.7(b) and 4.7(c).

These DFAs can be merged together to construct one single DFA, that accepts the same languages accepted by the three different DFAs. The distinction made between the different languages is made by the depth of the level of traversal.

Figure 4.8 shows an DFA that accepts ATREs of length 3, length 4 and length 5 respectively. The ATREs accepted, contain exactly one mismatch. If an edge is labeled by a mismatch character then the state it enters is labeled by `M`.

It is apparent from Figure 4.8 that if any t_j, a candidate TRE of the motif `ACG`, is submitted that contains exactly one mismatch, then an accept state is reached on level 3 of $Mis(ACG)$. Thus if the motif, $\rho = ACG$ and $t_j \in$

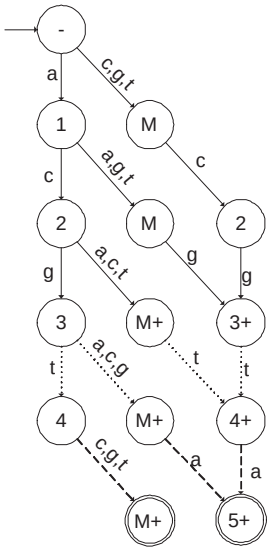


Figure 4.8: A DFA accepting ATREs of length 3, length 4 and length 5 that contains one mismatch

$\{AAG, AGG, ATG, ACT, ACA, ACC, CCG, GCG, TCG\}$ then only the solid edges of $Mis(ACG)$ will be followed to reach a final state on level 3 of $Mis(ACG)$.

However, the DFA in Figure 4.8 also accepts ATREs of length 4, as well as ATREs of length 5 that contain exactly one mismatch. In the case of candidate TREs of length 4 the applicable solid edges and nodes are followed up to depth 3. From depth 3 the dotted lines are followed until a final state is reached on level 4. Thus it is clear that if $Mis(ACGT)$ and
$t_j \in \{ACGA, ACGC, ACGG, ACCT, ACAT, ACTT, AGGT, ATGT,$
$AAGT, CCGT, GCGT, TCGT\}$
then an accept stage will be reached on level 4. Candidate ATREs of length 5 that contain exactly one mismatch will be accepted by following the applicable solid edges up to level 3, the dotted edges to level 4, and finally the applicable dashed edges that lead to a final state on level 5.

In a similar manner DFAs that accept
$Del(ACG)$,
$Del(ACGT)$,
$Del(ACGTA)$,
$Ins(ACG)$,
$Ins(ACGT)$, and
$Ins(ACGTA)$
can be constructed.

DFAs very similar to the DFA in Figure 4.8 can be constructed to accept TREs that contain both two mismatches or any other two allowed motif errors. These

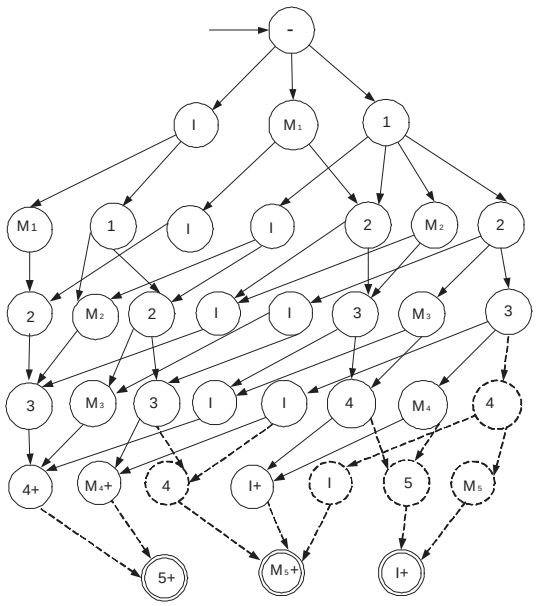


Figure 4.9: A DFA accepting ATREs of length 4 and length 5 that contain one insertion as well as one mismatch each.

DFAs are referred to as:
$MisMis(ACGT)$,
$MisDel(ACGT)$,
$DelDel(ACGT)$,
$DelIns(ACGT)$, and
$InsIns(ACGT)$.
For illustrative purposes, the structure of $MisIns(\rho)$ for some unspecified ρ, is shown in Figure 4.9.

For each loop of the program, the corresponding DFAs are constructed only once, and then subsequently decorated every time that TRs with a new motif are sought. $FA(\rho)$ has been explained as part of the presentation of $\mathsf{Fire}\mu\mathsf{Sat}_1$. However, in the case of $\mathsf{Fire}\mu\mathsf{Sat}_2$ $FA(\rho)$ is replaced by a set of DFAs, referred to as $FASet_\rho$. Each DFA in $FASet_\rho$ can be associated with a matcher function. The respective DFAs and matcher functions associated with these DFAs are represented in Table 4.2.

The original principles as explained for $FA(\rho)$ also hold for each element of this set. Note also that in order to meet the conditions regarding the order of precedence, the DFAs and their associated matcher functions are called in the order given in Section 4.3.1. The order of precedence is reflected in Table 4.2.

Moreover, the overall structure of the combined DFAs remains the same, irrespective of the motif that is used. Thus, the same structure as Figure 4.8 could

Types of TREs	Matcher function
PTRE	`ptre()`
1 mismatch	`mis()`
1 deletion	`del()`
1 insertion	`ins()`
2 mismatches	`mis2()`
1 mismatch & 1 deletion	`misDel()`
1 mismatch & 1 insertion	`misIns()`
2 deletions	`del2()`
1 deletion & 1 insertion	`delIns()`
2 insertions	`ins2()`

Table 4.2: TRE Types and Matcher functions

be used for motifs of say `TGA,TGAG,TGAGC`, etc.

In a similar manner, a single DFA structure that jointly represents three deletion DFAs, say
$DFA(Del(\texttt{ACG}))$,
$DFA(Del(\texttt{ACGT}))$, and
$DFA(Del(\texttt{ACGTA}))$
could be set up. This could also be done for three insertion DFAs, say
$DFA(Ins(\texttt{ACG}))$,
$DFA(Ins(\texttt{ACGT}))$, and
$DFA(Ins(\texttt{ACGTA}))$.
Once more, it can easily be verified that the overall structure is independent of the actual motif content.

Indeed, these observations hold for each of the 10 cases in Table 4.2, namely:

- The overall structure of a DFA matcher for a given error (or error combination), is independent of the actual content of the motif to be examined.
- The structure needed for the longest motifs embeds the structure for motifs of shorter length.

As a consequence, FireμSat$_2$ relies on this collection of ten pre-computed data structures corresponding to the ten TRE types in Table 4.2. Given a motif, ρ, the algorithm appropriately decorates all the data structures in this collection, and then uses the corresponding matcher functions listed in the table to identify TREs.

FireμSat$_2$ requires input parameters similar to that of FireμSat$_1$ described in Section 4.4.2.

In this outline the following functions are assumed:

- $build(l, \varepsilon)$ retrieves $FASet$, the set of pre-computed unlabeled DFA pattern matching structures, where l indicates the motif length and ε indicates the number of motif errors allowed. The $build$ function is similar to $genFA(l, \varepsilon)$ of FireμSat$_1$.
- $genMotifs(l)$ generates a set of all words from the alphabet $\Sigma = \{\texttt{A,C,G,T}\}$ of the length, (l), specified by the user.
- $decorate(FASet, \rho)$ returns $FASet_\rho$, a set of decorated pattern matching DFAs — i.e. all elements of $FASet$ are now labeled according to the current motif ρ under examination.
- $find(s, pos, \rho)$ returns the index of the start of the next candidate TR of motif ρ in the suffix of sequence s that starts at pos. It returns -1 if no match is found.

An outline of the FireμSat$_2$ algorithm is given below. The algorithm returns a set of tuples in the GCL below. Identical to FireμSat$_1$, the first component of each tuple denotes the start of a TR, the second denotes its length and the remaining components denote the number of PTREs, mismatches, deletions and insertions respectively.

Algorithm 4.4.5
func $Fire\mu Sat_2(l_{min}, l_{max}, \varepsilon, \tau, \alpha, \beta, p_m, p_d, p_i, S) : tuples$
Pre : $\{(0 < l_{min} \leq l_{max}) \wedge (0 \leq \varepsilon) \wedge (\sigma \leq \tau) \wedge (0 \leq tn_{atreC} \leq \alpha)$
$\wedge (\forall s \in S : length(s) > l) \wedge (0 \leq \beta \leq tn_{tre}) \wedge (S \in \Sigma^*)\}$
Post : $\{(pos, len) \in tuples \equiv \exists \rho : \Sigma^* \cdot |\rho| \in [l_{min}, l_{max}]$
$\wedge \forall s \in S : (\forall i \in [pos, pos + len) : \neg isEnd(s[i], \varepsilon, \tau, \alpha, \beta, p_m, p_d, p_i, \rho))$
$\wedge \forall s \in S : (\neg isEnd(s[pos + len], \varepsilon, \tau, \alpha, \beta, p_m, p_d, p_i, \rho))\}$
var i, ρ, pos, len, s

$tuples := \emptyset$
; **for** $l \in [l_{min}, l_{max}] \rightarrow$
$FASet := build(l, \varepsilon)$
$; W := genMotifs(l)$
for $s \in S \rightarrow$
for $\rho \in W \rightarrow$
$FASet_\rho := decorate(FASet, \rho)$
$; pos := find(s, 0, \rho)$
; **do** $(pos \neq -1) \rightarrow$
$< pos, len, n_{ptre}, n_m, n_d, n_i > :=$

$computeTR2(s, pos, FASet_\rho, \tau, \alpha, \beta, p_m, p_d, p_i)$
;**if** $(len > |\rho|) \rightarrow$
$tuples := tuples \cup \{<pos, len, n_{ptre}, n_m, n_d, n_i>\}$
[] $(len \leq |\rho|) \rightarrow$
$len := 0$
fi
; $pos := find(s, pos + len + 1, \rho)$
od
rof
rof
rof
;**return** *tuples*
cnuf

The algorithm $computeTR2$ (4.4.6) is called by $FireμSat2$ with pos indicating the starting position in s where a motif ρ has been found. The purpose of this function is to identify a single TR that has this motif, as well as its associated counters. The function uses $FASet_\rho$ to do this, as well as the specified threshold values τ, α and β and the penalty values p_m, p_d and p_i. It returns a five-tuple giving the length of the TR commencing at pos, as well as counts of the number of PTREs, mismatches, insertions and deletions, respectively.

An outline of the logic which $computeTR2$ follows is provided below in Algorithm 4.4.6. The following assumptions and conventions apply:

- The formal parameters of the function include a reference to the set of decorated pattern matching DFAs, $FASet_\rho$. This is to highlight the fact that all pattern matcher functions invoked in this function have access to the collection, even though this is not explicitly shown.
- The pattern matcher functions of Table 4.2 are invoked to determine whether the next TRE is of a given type. Because the logic of the function is given in GCL, and because the **if**-command in GCL is non-determinstic, each condition in the associated guarded command has to reflect the precedences relationship present in the ordering of Table 4.2.
- Each guarded command in the **if**-command appropriately updates the various counters. In addition, a variable c, is set to reflect the offset from $|\rho|$ that has to be used in order to compute the next starting position in s, from which to do the next TRE search.
- Additionally, if no TRE is found, a flag, *end* is set to terminate the search loop.

- A boolean function, *threshold*(), is assumed in the condition of the search loop. It determines whether the current value of the counters relative to the various threshold parameters indicate that the search should terminate. For simplicity its various parameters are not listed.

- the parameter t_j represents any candidate TRE relevant to the current motif ρ. Thus t_j represents a number of substrings of s. The length and the extraction of the substring t_j can be manipulated as follows:

  - $t_j - 2$: $|t_j - 2| = |\rho| - 2$,
  - $t_j - 1$: $|t_j - 1| = |\rho| - 1$,
  - t_j: $|t_j| = |\rho|$,
  - $t_j + 1$: $|t_j + 1| = |\rho| + 1$, and
  - $t_j + 2$: $|t_j + 1| = |\rho| + 2$ in order to obtain the substring of s of the correct length.

The function $ComputeTR2$ is partially specified in Algorithm 4.4.6.

Algorithm 4.4.6
func $computeTR2(s, pos, FASet_\rho, \tau, \alpha, \beta, p_m, p_d, p_i) : tuple$
$; n_m, n_d, n_i, tn_{ptre}, tn_{atre}, tn_{atreC}, := 0, 0, 0, 0, 1, 0$
$; trpos, pos, back := pos, pos + |\rho|, |\rho|$
$; t_j := nextCandidateTRE(s, pos)$
; **do** $(\neg isEnd(FASet_\rho, t_j, \tau, \alpha, \beta, p_m, p_d, p_i, n_m, n_d, n_i, tn_ptre, \beta, \alpha, back) \wedge$
$\neg s(EOS)) \rightarrow$
if $isPTRE(FASet_\rho, t_j) \rightarrow$
$tn_{ptre}, tn_{atreC} := tn_{ptre} + 1, 0$
$[\!]$ $(\neg isPTRE(FASet_\rho, t_j) \wedge isMis(FASet_\rho, t_j)) \rightarrow$
$tn_{atre}, tn_{atreC}, n_m := tn_{atre} + 1, tn_{atreC} + 1, n_m + 1$
$[\!]$ $(\neg isPTRE(FASet_\rho, t_j) \wedge \neg isMis(FASet_\rho, t_j) \wedge$
$isDel(FASet_\rho, t_j - 1)) \rightarrow$
$tn_{atre}, tn_{atreC}, n_d, := tn_{atre} + 1, tn_{atreC} + 1, n_d + 1$
$[\!]$ $(\neg isPTRE(FASet_\rho, t_j) \wedge \neg isMis(FASet_\rho, t_j) \wedge$
$\neg isDel(FASet_\rho, t_j - 1) \wedge isIns(FASet_\rho, t_j + 1)) \rightarrow$
$tn_{atre}, tn_{atreC}, n_i, := tn_{atre} + 1, tn_{atreC} + 1, n_i + 1$
$[\!]$ $(\neg isPTRE(FASet_\rho, t_j) \wedge \neg isMis(FASet_\rho, t_j) \wedge$
$\neg isDel(FASet_\rho, t_j - 1) \wedge \neg isIns(FASet_\rho, t_j + 1) \wedge$
$isMisMis(FASet_\rho, t_j)) \rightarrow$
$tn_{atre}, tn_{atreC}, n_m, := tn_{atre} + 1, tn_{atreC} + 1, n_m + 2$
$[\!]$ $(\neg isPTRE(FASet_\rho, t_j) \wedge \neg isMis(FASet_\rho, t_j) \wedge$
$\neg isDel(FASet_\rho, t_j - 1) \wedge \neg isIns(FASet_\rho, t_j + 1) \wedge$

$\neg isMisMis(FASet_\rho, t_j) \wedge isMisDel(FASet_\rho, t_j - 1)) \rightarrow$
$tn_{atre}, tn_{atreC}, n_d, n_m := tn_{atre} + 1, tn_{atreC} + 1, n_d + 1, n_m + 1$
$[\!]$ $(\neg isPTRE(FASet_\rho, t_j) \wedge \neg isMis(FASet_\rho, t_j) \wedge$
$\neg isDel(FASet_\rho, t_j - 1) \wedge \neg isIns(FASet_\rho, t_j + 1) \wedge$
$\neg isMisMis(FASet_\rho, t_j) \wedge \neg isMisDel(FASet_\rho, t_j - 1) \wedge$
$isMisIns(FASet_\rho, t_j + 1)) \rightarrow$
$tn_{atre}, tn_{atreC}, n_i, n_m := tn_{atre} + 1, tn_{atreC} + 1, n_i + 1, n_m + 1$
$[\!]$ $(\neg isPTRE(FASet_\rho, t_j) \wedge \neg isMis(FASet_\rho, t_j) \wedge$
$\neg isDel(FASet_\rho, t_j - 1) \wedge \neg isIns(FASet_\rho, t_j + 1) \wedge$
$\neg isMisMis(FASet_\rho, t_j) \wedge \neg isMisDel(FASet_\rho, t_j - 1) \wedge$
$\neg isMisIns(FASet_\rho, t_j + 1) \wedge isDelDel(FASet_\rho, t_j - 2)) \rightarrow$
$tn_{atre}, tn_{atreC}, n_d := tn_{atre} + 1, tn_{atreC} + 1, n_d + 2$
$[\!]$ $(\neg isPTRE(FASet_\rho, t_j) \wedge \neg isMis(FASet_\rho, t_j) \wedge$
$\neg isDel(FASet_\rho, t_j - 1) \wedge \neg isIns(FASet_\rho, t_j + 1) \wedge$
$\neg isMisMis(FASet_\rho, t_j) \wedge \neg isMisDel(FASet_\rho, t_j - 1) \wedge$
$\neg isMisIns(FASet_\rho, t_j + 1) \wedge \neg isDelDel(FASet_\rho, t_j - 2) \wedge$
$isDelIns(FASet_\rho, t_j)) \rightarrow$
$tn_{atre}, tn_{atreC}, n_d, n_i := tn_{atre} + 1, tn_{atreC} + 1, n_d + 1, n_i + 1$
$[\!]$ $(\neg isPTRE(FASet_\rho, t_j) \wedge \neg isMis(FASet_\rho, t_j) \wedge$
$\neg isDel(FASet_\rho, t_j - 1) \wedge \neg isIns(FASet_\rho, t_j + 1) \wedge$
$\neg isMisMis(FASet_\rho, t_j) \wedge \neg isMisDel(FASet_\rho, t_j - 1) \wedge$
$\neg isMisIns(FASet_\rho, t_j + 1) \wedge \neg isDelDel(FASet_\rho, t_j - 2) \wedge$
$\neg isDelIns(FASet_\rho, t_j) \wedge isInsIns(FASet_\rho, t_j + 2)) \rightarrow$
$tn_{atre}, tn_{atreC}, n_i, n_m := tn_{atre} + 1, tn_{atreC} + 1, n_i + 1, n_m + 1$
$[\!]$ $(\neg isPTRE(FASet_\rho, t_j) \wedge \neg isMis(FASet_\rho, t_j) \wedge$
$\neg isDel(FASet_\rho, t_j - 1) \wedge \neg isIns(FASet_\rho, t_j + 1) \wedge$
$\neg isMisMis(FASet_\rho, t_j) \wedge \neg isMisDel(FASet_\rho, t_j - 1) \wedge$
$\neg isMisIns(FASet_\rho, t_j + 1) \wedge \neg isDelDel(FASet_\rho, t_j - 2) \wedge$
$\neg isDelIns(FASet_\rho, t_j) \wedge \neg isInsIns(FASet_\rho, t_j + 2)) \rightarrow skip$

fi
$; pos := pos + |t_j|$
$; t_j := nextCandidateTRE(s, pos)$

od
$; len := pos - trpos - back$
; **return** $< trpos, len, n_{ptre}, n_m, n_d, n_i >$
cnuf

4.4.4 Theory Underlying FireμSat$_3$

FireμSat$_3$ is distinct from both FireμSat$_1$ and FireμSat$_2$ in the sense that it traverses the input string only once for all introductory motifs of a certain length. Thus if 3 has been allocated to l_{min} and 5 to l_{max}[2] then there will only be three traversals of the genetic input string.

If $l_{min} = 3$ and $l_{max} = 5$ then FireμSat$_3$ constructs three generic FAs — one for each PTRE of length 3, 4 and 5 respectively. For each traversal (thus for each selected motif length), the parameter *pos* is set equal to the first character of the input string during the first decoration of a DFA of a certain length. Consider FA_ρ of FireμSat$_3$ where $|\rho| = 3$. The generic FA_ρ reads the genetic input string and uses three consecutive characters, starting at *pos* as the introductory motif, ρ. Thereafter *pos* becomes either the position after the last detected TR or if no TR is found, then *pos* is incremented by 1 until it becomes the starting position of a TR.

From section 4.4.1 it will be recalled that the output of FireμSat$_3$ differs from that of both FireμSat$_1$ and FireμSat$_2$, in the sense that each TR per motif length is only reported on once and duplicate data is avoided.

Consider the TR `ACTACTACTA` within a genetic input string. This TR will be detected only once by FireμSat$_3$ as a TR where the PTRE is equal to `ACT`. However, FireμSat$_1$ and FireμSat$_2$ will detect and report on the three different TRs in the case of motif length 3. These three different TRs will be identified, namely as having `ACT`, `CTA` and `TAC` as PTREs and will be found during three respective traversals.

Algorithm 4.4.7
func $Fire\mu Sat_3(l_{min}, l_{max}, \varepsilon, \tau, \alpha, \beta, p_m, p_d, p_i, S) : tuples$
Pre : $\{(0 < l_{min} \leq l_{max}) \wedge (0 \leq \varepsilon) \wedge (\sigma \leq \tau) \wedge (0 \leq tn_{atreC} \leq \alpha)$
$\wedge (\forall s \in S : length(s) > l) \wedge (0 \leq \beta \leq tn_{tre}) \wedge (S \in \Sigma^*)\}$
Post : $\{(pos, len) \in tuples \equiv \exists \rho : \Sigma^* \cdot |\rho| \in [l_{min}, l_{max}]$
$\wedge \forall s \in S : (\forall i \in [pos, pos + len) : \neg isEnd(s[i], \varepsilon, \tau, \alpha, \beta, p_m, p_d, p_i, \rho))$
$\wedge \forall s \in S : (\neg isEnd(s[pos + len], \varepsilon, \tau, \alpha, \beta, p_m, p_d, p_i, \rho))\}$
var i, ρ, pos, len, s

$tuples := \emptyset$
; **for** $l \in [l_{min}, l_{max}] \rightarrow$
$FA := genFA(l, \varepsilon)$
{ *W, the set of motifs is not necessary to generate because the algorithm discovers the motifs directly from genetic sequence set S* }

[2]These parameters are described in Section 4.4.2.

for $s \in S \rightarrow$
$pos, \rho := 0, s[0, l)$
$; FA_\rho := decorate(FA, \rho)$
; **do** $(pos \neq -1) \rightarrow$
$< pos, len, n_{ptre}, n_m, n_d, n_i > :=$
$computeTR1(s, pos, FA_(\rho), \tau, \alpha, \beta, p_m, p_d, p_i)$
; **if** $(len > |\rho|) \rightarrow$
$tuples := tuples \ \cup \{< pos, len, n_{ptre}, n_m, n_d, n_i >\}$
$[\!]$ $(len \leq |\rho|) \rightarrow$
$len := 0$
fi
$; pos, \rho := pos + len + 1, s[pos, pos + l)$
$; FA_\rho := decorate(FA, \rho)$
od
rof
rof
; **return** *tuples*
cnuf

4.5 Conclusion

Before Chapter 4 introduced the three FIREμSAT algorithms, the underlying FA theory which forms the basis for the specifications of the 3 implementations — FIREμSAT$_1$, FIREμSAT$_2$ and FIREμSAT$_3$ have been provided in this chapter. The input parameters of FIREμSAT$_1$ and FIREμSAT$_2$ are similar. FIREμSAT$_3$ has only been implemented to search for microsatellites with a motif length of three. The objective of the implementation of FIREμSAT$_3$ is threefold. Firstly it is of theoretical and practical relevance in the sense that it shows that it is possible to implement FIREμSAT$_3$. Secondly the opportunity is provided to compare the data generated by FIREμSAT$_3$ to the data generated by FIREμSAT$_1$ and FIREμSAT$_2$. Thirdly the runtime of FIREμSAT$_1$ and FIREμSAT$_2$ can be compared to the runtime of FIREμSAT$_3$. GCL has been used to give the algorithmic details of each of the three algorithms. A comparison of the runtime of the different FIREμSAT algorithms is presented in Table 4.3. These data files are discussed in detail in Chapter 2 and Chapter 6. Details of the hardware used during the trial runs can be found in Chapter 6. From Table 4.3 it is evident that the FA construction of FIREμSAT$_1$ takes 334.9s while the duration for the actual traversing of the 9kB file is a fraction of a second.

The runtime of FIREμSAT$_1$ is slower than that of FIREμSAT$_2$ and FIREμSAT$_3$. Mainly because the construction of the different FAs is time consuming. If the general

Software	Swam.txt (9kB)	Fusarium Graminearum.txt (33MB)
FireμSat$_1$	335s	$<$ 360s
FireμSat$_2$	$<$ 1s	90s
FireμSat$_3$	$<$ 1s	20s

Table 4.3: A runtime comparison between the three FireμSat implementations on the swam50.txt file (9k) and on Fusarium Graminearum.txt (33MB)

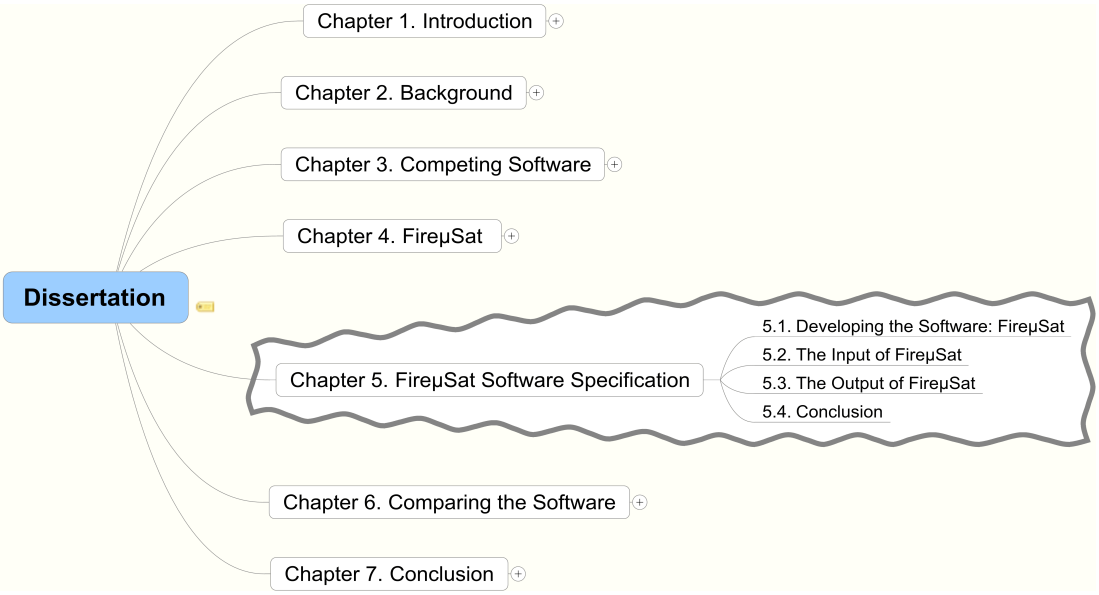
FAs can be constructed and stored in memory then the runtime of FireμSat$_1$ will definitely be reduced. FireμSat$_1$ is easily extendible. Future research will entail the extension of FireμSat$_1$ to search for minisatellites too. FireμSat$_2$ is fast but not easy to extend. FireμSat$_3$ is faster than both FireμSat$_1$ and FireμSat$_2$. Future research initiatives will also include to investigate how to merge the theoretical approaches of FireμSat$_1$ and FireμSat$_3$. FireμSat$_1$ and FireμSat$_2$ generate duplicate data that is not generated by FireμSat$_3$. The developments of FireμSat$_1$, FireμSat$_2$ and FireμSat$_3$ were independent. The three algorithms generate the same results. This independent cross validating development confirms the accuracy for all three algorithms.

The details of the input requirements and of the output of FireμSat are presented in Chapter 5, in a similar manner to the implementation details of Tandem Repeats Finder and STAR in Chapter 3. The results of the three implementations (Tandem Repeats Finder, STAR and FireμSat$_2$) are compared in Chapter 6 in terms of the criteria presented in Chapter 2, Section 2.6. As mentioned in Chapter 1, a brief comparison between these implementations and IMEx [Mudunuri & Nagarajaram (2007)] will also be included.

Chapter 5 will elaborate on the input and output details of FireμSat. The ways in which the criteria listed in Section 2.6 have been met by Tandem Repeats Finder, STAR and FireμSat will be discussed in Chapter 6. The FireμSat software is included on the compact disc that accompanies this dissertation.

Chapter 5

FireμSat Software Specification



In Chapter 2, a literature overview of algorithms that aim to detect tandem repeats on DNA was provided. Two algorithms that deal in an effective manner with the detection of microsatellites (as defined in Chapter 2, Section 2.2) on DNA were identified, namely Tandem Repeats Finder [Benson (1999)] and STAR (Search for Tandem Approximate Repeats) [Delgrange & Rivals (2004b)]. "Effective" in the context of this dissertation, firstly implies an algorithm that detects, in a single run, microsatellites that contain substitutions, deletions, as well as insertions. It also implies that the relevant algorithm has a running time smaller or equal to $O(np + nlogn)$, where n is the length of the genetic sequence

file under investigation and p is the length of the motif to be detected. Similarly to FireμSat (the proposed algorithms), STAR detects microsatellites only, whereas Tandem Repeats Finder detects microsatellites, as well as minisatellites (defined in Chapter 2, Section 2.2) and satellites (defined in Chapter 2, Section 2.2).

Because the required inputs to both Tandem Repeats Finder (see Chapter 3, Section 3.1) and to STAR (see Chapter 3, Section 3.2) are not trivial, the matter was given detailed attention in Chapter 3. In Chapter 4 the theoretical background as well as the specifications of the algorithm proposed in this dissertation, FireμSat, has been presented. In Chapter 5 the input and output of FireμSat are discussed in a similar manner to the discussion of that of Tandem Repeats Finder and STAR in Chapter 3.

5.1 Developing the Software: FireμSat

FireμSat has *inter alia* been implemented by Mr T.R. Fourie (FireμSat$_2$) and Mr A.P.F. Marais (FireμSat$_1$) as partial fulfillment of their B.Sc. honours degrees. Mr T.R. Fourie implemented FireμSat$_2$ in C++. Mr A.P.F. Marais implemented FireμSat$_1$ by writing some of the code in Python and some of the code in C++.

Additionally for the sake of theoretical interest Mr P.V. Reyneke implemented FireμSat$_3$.

The FireμSat software included on the compact disc, that accompanies this dissertation runs from the DOS shell under a Windows operating system. Development is in progress to create a full GUI (graphical user interface) for FireμSat that complies to human computer interaction (HCI) prescriptions. At present a simple implementation of the GUI is implemented for FireμSat$_2$. FireμSat$_2$ along with its GUI shell can be accessed/downloaded online at *www.dna-algo.co.za*.

The mentioned simple GUI presented in Figure 5.1 complies to minimum specifications of the author of this dissertation. The current GUI is created in such a manner that it illustrates the unique computability features of the FireμSat software that contribute to its useability.

5.2 The GUI Input of FireμSat

After the user has double clicked on the FireμSat icon that represents the installed FireμSat software, the Windows GUI included in Figure 5.1 will appear.

The GUI contains different edit boxes labeled with appropriate names.

A description of the different edit boxes follows:

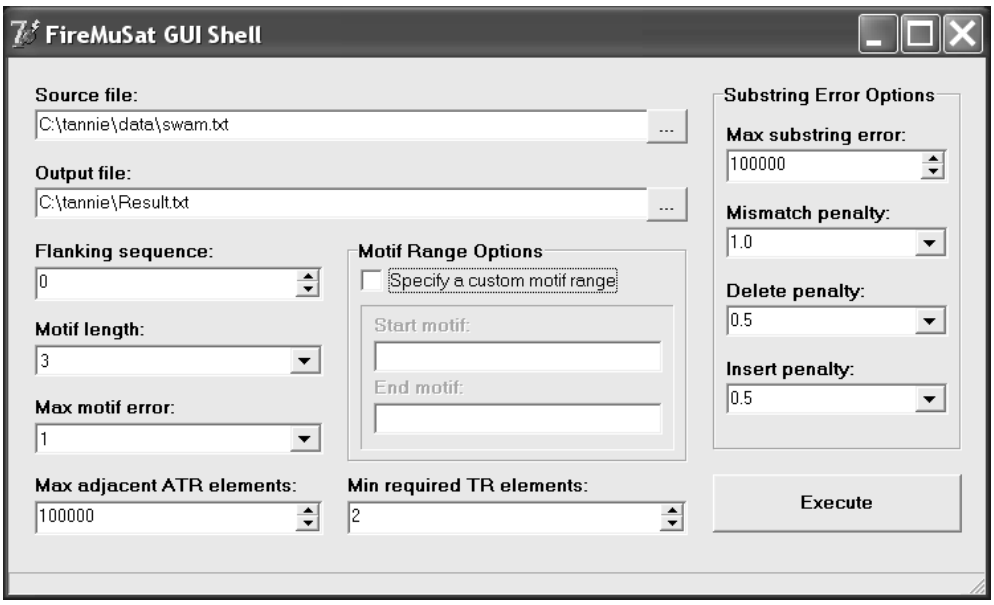


Figure 5.1: The GUI of the FireμSat software

- Source File
  The user should enter in this edit box the path to the genetic file (in FASTA format), which is to be scanned for TRs. If the user enters an incorrect path and/or file name, then FireμSat will display the following error message:

  ```
  Source file not found!
  ```

- Output File
  The user should specify the path to a directory,s where the output file is to be created. The output file can be specified to be written in either a .txt (text) format or a .csv (comma-separated value) format. In order to enhance the useability of the output results, it is recommended that the user specifies output to be written in .csv format rather than .txt format. The .txt files can be opened by either Wordpad or Notepad while the .csv files can also easily be opened by any spreadsheet program — for example in Open Office, Excel, or Borland's Quattro.

  The path name to indicate where FireμSat should write the output file should be specified. If an invalid path name is specified, then FireμSat will display an error message. For example if the faulty path has been entered as:

  ```
  C:\FireMuSatX\Result.txt.

  Could not create output file:C:\FireMuSatX\Result.txt.
  ```

- Flanking Sequence
  The flanking sequence the user requires, should be specified in this edit box. Recall from Chapter 3 that the flanking sequence is a certain number of nucleotides that are output before and after a detected TR. The contents of the flanking sequence edit box, can be adjusted by means of a spin control on the right of the edit box. FireμSat displays an error message:

  ```
  Value "-5" supplied to argument "-padding"
  is invalid.
  ```

  if the negative number "-5" has been input. The *Flanking Sequence* edit box has been selected as a numerical edit box — only numerical characters can be typed in it.

- Motif Length
  Thus in the edit box *Motif Length* the length of the motif or PTRE, of which TRs should be found should be specified. The value of the *Motif Length* edit box can be manipulated by the combo box on the right of the *Motif Length* edit box. The combo box presents a selection of valid parameters. Therefore no error messages are needed regarding the *Motif Length* edit box.

- Max Motif Error
  Thus the motif error indicates the number of mutations the user chooses to allow on a motif of a preselected length. The number of mutations allowed can be manipulated by clicking on the combo box on the right of the edit box *Max Motif Error*. The type of mutations, as well as the limit on the number of mutations allowed, are discussed in detail in Chapter 4, Section 4.3. The combo box presents a selection of valid parameters, corresponding to the selected motif length. Therefore, no error messages are needed regarding the *Max Motif Error* edit box.

- Max adjacent ATR elements
  The edit box *Max adjacent ATR elements* provides to the users the option of entering a value that indicates the maximum number of ATREs (approximate tandem repeat elements) that are allowed to occur next to each other. The user can manipulate the number of ATREs that occur adjacently by the spin control on the right of the edit box *Max adjacent ATR elements*. Chapter 4, Section4.3.2 provides the theoretical detail regarding the maximum number of consecutive ATR elements that are allowed. If a negative number is entered by the user in the *Max adjacent ATR elements* edit box, then the number entered is by default converted to zero.

- Motif range option
  The motif range option can be activated by clicking on the check box that appears beneath the correspondingly labeled text box.

  If this option is activated then the *Start motif* and *End motif* turns to black instead of grey as it was before the option had been activated. The motif range option provides the possibility to specify a range of motifs FireμSat should search for.

  However, the motifs should all be of the same length that corresponds to the length specified in the edit box *Motif Length*. If for example only TRs of one certain introductory PTRE or motif is required, then the start and end motif should be specified as the same.

  The start motif should always be lexicographically smaller or equal to the end motif. If the start motif is not lexicographically smaller or equal than the end motif, then FireμSat will display an error message:

  ```
  End motif "AAA" is lexicographically smaller than
  the start motif "AAC".
  ```

- Min required TR elements
  To avoid the output of unwanted data, the minimum number of TREs that have to be detected before a TR is output,s may be indicated.

  The edit box *Min required TR elements* can be set by using the spin control on the right of it.

- Substring Error Options
  It can be recalled from Chapter 4 that the *substring error* is a measure of the extent to which the number (weighted as described below) of ATREs in the candidate TR, exceeds the number of PTREs. The measure is computed at appropriate points by FireμSat and then compared against a user-specified threshold value indicated by the user in the edit box *Max substring error*. Currently there are two versions of FireμSat$_2$ available. The one allows the calculating of the relative substring error; the other the calculating of the absolute substring error. During processing, the calculated substring error, should always be smaller than the substring error indicated by the user in the edit box *Max substring error*. The value of the maximum substring error allowed, can be modified by using the spin control on the right of the edit box *Max substring error*. Error control is done by FireμSat. If a negative value is entered as the maximum substring error, then FireμSat automatically converts the value to zero.

  In line with the guidelines suggested by Benson (1999), the value of the substring error depends, *inter alia* on penalties (or weights) allocated by

the user to mismatches, deletions and insertions respectively. Underneath the edit box *Max substring error* are three additional edit boxes, where the values of the different penalties can be adjusted according to the requirements of the user. The *Mismatch penalty* edit box, the *Deletion penalty* edit box, as well as the *Insertion penalty* edit box have combo boxes on their right that serve to modify the three different penalties respectively.

There may be relied on system default values for the penalties. The system default values appear automatically in the three penalty edit boxes. The default values in the respective edit boxes are as follows:

- *Mismatch penalty:* 0.5.
- *Delete penalty:* 1.0.
- *Insert penalty:* 1.0.

A penalty weight of 0 may be chosen for one or more of the mutation types, in which case no penalty is assigned to ATREs that derive from that mutation type. The range of the penalty values is ≥ 0 and ≤ 1 If an invalid value is entered as a penalty value, for example 1.45, then FireμSat displays:

```
Value:"01.45" supplied to argument "p_m" is out of range.
```

- Execute
  Finally, the *Execute* button appear at the bottom of the window. After the *Execute* button has been clicked on, FireμSat will create an output file of either type .txt or type .csv in the specified directory. The output generated by FireμSat will be discussed in the Section 5.4 of this chapter.

5.3 The Commandline Input Specification of FireμSat

All the functionality mentioned for the GUI (and more) can be accessed when running FireμSat from the commandline. The commandline specification, terms and ranges are defined in Section 5.2, for FireμSat is:

```
 FireMuSat.exe
 -source-file=file_in.fasta.txt
 -output-file=file_out.csv.txt
 -motif-len=3
[-start-motif=AAA]
[-end-motif=AAA]
[-max-motif-err(=inf)]
```

motif	pos	len	TR	n_ptre	n_atre	n_m	n_d	n_i
GAT	6495	15	GATAATTATAATTAT	1	4	4	0	0
GCT	4507	15	GCTCCTGCAGCTGCG	2	3	3	0	0
GGT	7921	18	GGTGCTGGTGCTGGTAGT	3	3	3	0	0
GTA	6138	29	GTAGGTAGTATTATACTAGGTAAGTAGGA	2	7	3	1	3
GTA	6142	25	GTAGTATTATACTAGGTAAGTAGGA	2	6	3	1	2

Table 5.1: Sample output generated by FireμSat$_2$

```
[-max-substr-err(=inf)]
[-p-m(=0.5)]
[-p-i(=1.0)]
[-p-d(=1.0)]
[-max-adj-atres(=inf)]
[-min-req-tres(=0)]
[-padding(=0)]
[-use-relative-substr-error(off)]
     - if this flag is set then relative error is used
[-max-gaplength(=0)]
     - join TR with gap as specified
[-alt-precedence(off)]
     - if this flag is set then deletions precedes mismatches
[-no-backward-search(off)]
     - if this flag is set then no backward search is done
```

The parameters in square brackets `[]` are optional. The default values, shown in round brackets `()`, are the preferred values.

Note, the last three optional parameters are undocumented and touch upon in Section 6.5). These parameters were added for testing and comparison purposes.

5.4 The Output of FireμSat

The output of FireμSat is a text file in comma separated value (.csv) file format. This implies that each data object is separated by an *eoln* character(s)(line feed and a carriage return character). The data fields within each data object are separated by commas. The file format as described corresponds to prerequisites of .csv extension files.

Therefore, if the user indicates the output file should be a .csv file, then the output file created by FireμSat will automatically open in a spreadsheet program if double clicked on the icon that represents the file.

In Table 5.1 is an example of five lines of output generated by FireμSat in .csv file format.

FireμSat generates nine different columns as output. The nine different columns have appropriate headers. The headers and a brief explanation of the column contents follow below:

- Column 1: `motif`
  The header of the first column is *motif*. In the context of the output of FireμSat the header *motif* indicates the introductory motif or PTRE of which "copies" are detected and reported on.

- Column 2: `pos`
  The header of the second column is *pos*. The second column stores the offset index position of the respective detected TRs as an integer. Within the context of the FireμSat output, offset refers to the number of nucleotides that precedes a TR, relative from the start of the input file.

- Column 3: `len`
  The header, *len*, of the third column refers to the word length; the third column displays the lengths of the respective, detected TRs as integer values. The length of a TR is equal to the number of nucleotides it consists of.

- Column 4: `TR`
  The fourth column has the header *TR*. In column 4 the different TRs that were detected are written out as a concatenation of alphabetical characters.

- Column 5: `n_ptre`
  The header of the fifth column n_ptre refers to the number (an integer) of TREs that are exact copies of the introductory motif within the respective detected TRs.

- Column 6: `n_atre`
  The sixth column output the number of ATREs that were counted within the respective detected TRs. Recall from Chapter 4, Section 4.3 that ATREs refer to TREs that are not exact copies of the introductory motif, but that ATREs are TREs with allowed mutations. The output is an integer value that represents the number of ATREs, that occur within the detected TR.

- Column 7: `n_m`
  The seventh column contains the number of mismatches that occurred within the respective detected TRs. A mismatch is one of the types of allowed mutations. Mutations including mismatches have been discussed in detail in Chapter 4, Section 4.3.

- Column 8: `n_d`
  The eighth column contains the number of TREs where deletions have occurred of the respective detected TRs. Allowed mutations are discussed in detail in Chapter 4, Section 4.3.

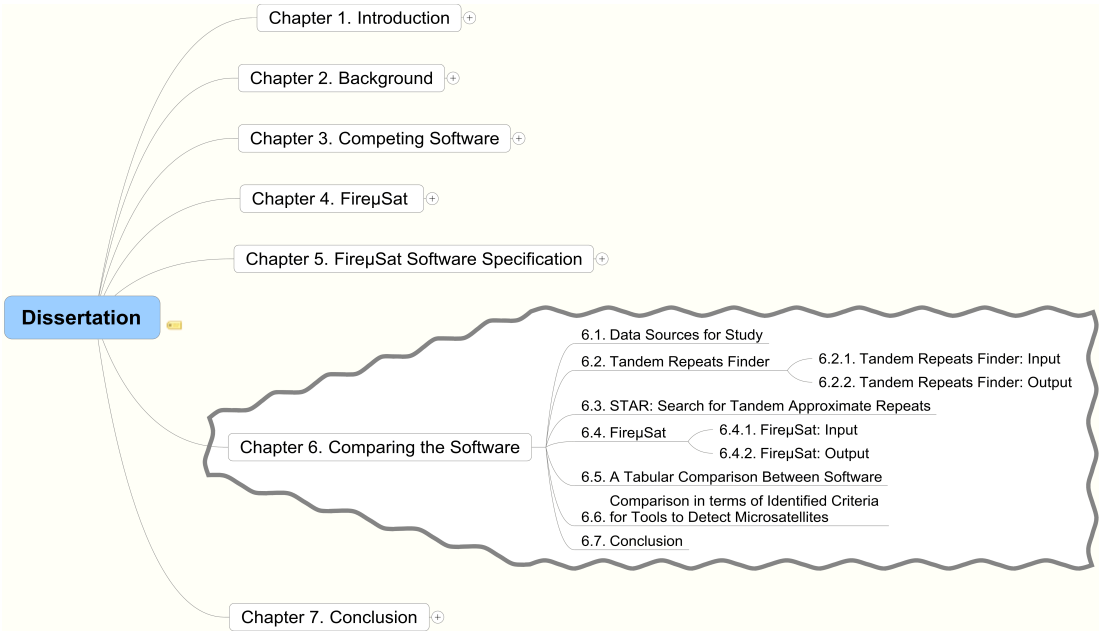
- Column 9: `n_i`
  The last column, column 9 indicates the number of TREs that contain insertions. An insertion is an allowed mutation type, discussed in detail in Chapter 4, Section 4.3.

5.5 Conclusion

The input and output of FireμSat have been discussed in this chapter. Chapter 6 is a comparison of the three different algorithms: Tandem Repeats Finder (discussed in Chapter 3), STAR (presented in Chapter 3) and FireμSat (introduced in Chapters 4 and 5). The discussion in Chapter 6 will refer to the criteria proposed in Chapter 2, Section 2.6.

Chapter 6

Comparing the Software



In Chapter 2, a literature overview of algorithms that aim to detect TRs on DNA was provided. Two algorithms that deal in an effective manner with the detection of microsatellites (as defined in Section 2.2) on DNA were identified namely, Tandem Repeats Finder [Benson (1999)] and STAR (Search for Tandem Approximate Repeats) [Delgrange & Rivals (2004b)]. "Effective" in the context of this dissertation, firstly implies algorithms that detect microsatellites containing substitutions, deletions, as well as insertions at once. It also implies that the relevant algorithms have a running time smaller or equal to $O(np + nlog(n))$, where n is

the length of the genetic sequence file under investigation and p is the length of the motif to be detected. Similar to FireμSat (the proposed algorithm) STAR detects microsatellites only, whereas Tandem Repeats Finder detects microsatellites, as well as minisatellites (defined in Section 2.2) and satellites (defined in Section 2.2).

The required input to Tandem Repeats Finder (Section 3.1) and to STAR (Section 3.2) is not trivial, therefore it was discussed in detail in Chapter 3. In Chapter 4 the theoretical background, as well as the specifications of the algorithm proposed in this dissertation, FireμSat, has been presented. In Chapter 5 the input and output of FireμSat are discussed in a similar manner to the discussion of that of Tandem Repeats Finder and STAR in Chapter 3. Chapters 3, 4 and 5, thus provide a basis for Chapter 6, in which a comparison is made among Tandem Repeats Finder, STAR and FireμSat in relation to the criteria presented in Chapter 2, Section 2.6. In Section 6.1 a brief overview of the data used for the comparative study is presented. Sections 6.2, 6.3 and 6.4 report on the performance of Tandem Repeats Finder, STAR and FireμSat respectively. Section 6.5 presents a tabular comparison followed by a comparison between FireμSat and IMEx specifically. Section 6.6 shows in which manner FireμSat complies to the criteria compiled in Chapter 2, Section 2.6. Section 6.7 concludes this chapter.

6.1 Data Sources for Study

As discussed in Section 2.4 two sets of data were used to compare these algorithms in terms of their practical execution. They are the following:

1. *Data of the fungus* ***Fusarium Graminearum***
   Data of this fungus was made available by the Center of Genome Research. It constitutes 33 930 392 bytes. The data is in FASTA format and thus consists of concatenated characters over the alphabet $\Sigma = \{A, C, G, T\}$. For the purposes of the first comparative study only the first 8400 base pairs were extracted. The extracted base pairs are stored in a file `swam.txt`. `swam.txt` can be found on the CD that accompanies this dissertation. In terms of run time performance the run time of the different algorithms were compared on both the extracted file of 8400 characters, as well as the complete sequenced *Fusarium Graminearum* genome. The base pairs of the *Fusarium Graminearum* genome has been divided in different so called scaffolds or attached regions. The number of base pairs varies for each scaffold. The second scaffold is labeled with the heading > *Fusarium graminearum 1.54 (scaffold 1)*. The data of the *Fusarium graminearum* genome is included on the CD that accompanies this dissertation.

2. *Data of the* ***Cylindrocladium Pauciramosum***
   The genomic data of the *Cylindrocladium Pauciramosum* was generated by Dr.Wright, a molecular biologist of the University of Pretoria. The Random Amplified Microsatellites (RAMS) method was applied in order to obtain the generated data[1]. The format of the data obtained is relevant to the present discussion. The data generated by the RAMS method consists of various sequences, each containing data of different so called "libraries" (explained in Chapter 2 (Section 2.4)) and having different sequence headers. The following gives examples of two of the genetic sequences generated by Wright reported on in Wright *et al.* (2007), these sequences are contained in the same file:

```
>BL141
CCACCACCACCACCAACACAATTGCACCGCTAGTGGCTATATTTGATGCC
CTCAAAATTCCCGCACCGTGGGCACCAGAGGCCAAGGATTTCGACTACGC
AAACACGACTTTGCTGATTATGGGTGGTGGATCCAGCACCGGCAAATTTG
GCGTACAATTAGCCAAGTTAGCAGGCATTGGCAAGATTGTTGTTGTTGTT
G
>BL204
ACAACAACAACAACAGTAGGAAGGAAATAATAACCATAGAAACCAGTTTT
TGAAATCTGTTCAGTTTAATTAATTTTGCAATTTTTATTCCCTATTTTTG
TCTAGAGCGTAGGAGTGTAAGTGGATCTCGCATCCTTCGAGAGCCCGCTG
CTCATTACGGAGACCCTCCCTCCACTCTTTGCTTGCTTCAACAAGGTTGA
ACTTTGTTGCGGCTGTCTCTCTCTAGTCTTTTGATGCCATCGCTCATTTT
GGCCTTTCCCGCGGTCTGTTGCCGCCTTCTCCTTATCTCCTGTGTGTGTG
TGTGTG
```

   For the purposes of the data comparison there will also be reported on the data file WrightSEQ2.txt. However, in the interest of avoiding duplication of information there will only be reported on the number of TRs detected during the trial run of WrightSEQ2.txt. The complete data file can be found on the CD that accompanies this dissertation.

In the following sections the outcome of trial runs using `swam.txt` will be discussed.

[1]The RAMS method is a fairly complicated technique and the details thereof is, as mentioned in Section 2.2, beyond the scope of this dissertation. A description of the technique can be found in Van der Nest *et al.* (2000).

6.2 Tandem Repeats Finder

6.2.1 Tandem Repeats Finder: Input

Tandem Repeats Finder is installed as *trf*. After typing *trf* at the command line the following output appears:

```
Please use: trf File Match Mismatch Delta PM PI MinScore MaxPeriod
```

Each word after *trf* in this message indicates an input parameter required by Tandem Repeats Finder. Each of these parameters were discussed in detail in Chapter 3. Forthcoming,s a brief overview is provided of the different parameters, as well as of the values assigned to them during the trial run of `swam.txt`.

1. `File`: The input DNA sequence file in FASTA format, swam.txt

   It is possible to recall from Chapter 3 that multiple genomic sequences with different headings in the same file are acceptable as long as the data is in FASTA format. The benefit thereof is that it enables the user to process a large sequence file consisting of multiple genomic sequences without creating very small files for each genetic sequence with their different headings.

2. `Match`, `Mismatch`, `Delta`: *Alignment parameters that represent the weights for matches, mismatches and indels respectively*

   These parameters are used for Smith-Waterman style local alignment wraparound dynamic programming [Benson (2003b)] as discussed in Chapter 3. Lower weights entered as the alignment parameters of Tandem Repeats Finder allow alignments with more mismatches and indels. `Match` = 2 has proven effective with `Mismatch` and `Delta` ranging between 3 and 7. Mismatch and indel weights are interpreted as negative numbers. The values options, 3 is more permissive and 7 is less permissive ($-3 > -7$) [Benson (2003b)]. The values 2, 3 and 5 for `Match`, `Mismatch` and `Delta`, have been entered respectively during the run of Tandem Repeats Finder on `swam.txt`.

3. `PM` *and* `PI`: *Detection Parameters*
   Detection parameters consist of a matching probability Pm and an indel probability Pi. $Pm = .80$ and $Pi = .10$ by default and cannot be altered in the revised, 2003 version of the program.

4. `Minscore`: *Minimum alignment score*
   The minimum alignment score indicates the alignment score that must be

met or that must be exceeded for a tandem repeat to be reported [Benson (2003b)]. The minimum alignment score has been discussed in more detail in Chapter 3. For the purposes of the comparison of the output of the different software the minimum alignment score has been chosen as 10. Thus the detected TR should constitute at least 5 nucleotides - the matching weight is 2 and $2 \times 5 = 10$

5. `Maxperiod`: *Maximum period size*
   In Tandem Repeats Finder, period size is the program's best guest at the length of the TREs that are detected within an identified TR. The parameter `Maxperiod` is discussed at length in Chapter 3. For the purposes of the presented comparison, `Maxperiod` has been chosen as 3, consequently Tandem Repeats Finder reports during the run under discussion all TRs detected that has a motif length smaller or equal to 3.

Tandem Repeats Finder provides three additional options (`-f`, `-m` and `-d`), that the user may specify as part of the command line input. These "switches" are included to provide the user with additional options in terms of output. These switches are discussed in Chapter 3 and have not been set during the trial run as it will not influence the number of TRs detected.

6.2.2 Tandem Repeats Finder: Output

Tandem Repeats Finder generates a summary table of repeats, as well as an alignment explanation as output. The summary table generated by Tandem Repeats Finder is as follows:

```
Tandem Repeats Finder Program writen by:

Gary Benson
Department of Biomathematical Sciences
Mount Sinai School of Medicine
Version 4.00


Sequence: swam


Parameters: 2 3 5 80 10 10 3


1931 1937 3 2.3 3 100 0 14 71 28 0 0 0.86 ACA ACAACAA
3587 3596 3 3.3 3 100 0 20 0 0 70 30 0.88 GTG GTGGTGGTGG
3672 3683 3 4.0 3 88 0 19 0 25 41 33 1.55 TGC TGGTGCTGCTGC
3771 3795 2 12.5 2 52 16 13 48 8 8 36 1.62 TA TATATAGCAATCTAAAATGATATAT
3802 3810 2 4.5 2 71 0 13 0 0 44 55 0.99 GT GTTTGTGTG
5859 5870 1 12.0 1 81 0 19 91 0 8 0 0.41 A AAAAAAAAGAAA
5993 6003 2 5.5 2 100 0 22 45 0 0 54 0.99 TA TATATATATAT
6195 6199 1 5.0 1 100 0 10 0 0 0 100 0.00 T TTTTT
6386 6395 3 3.3 3 100 0 20 30 40 0 30 1.57 CTA CTACTACTAC
6471 6483 2 7.0 2 83 16 21 53 0 0 46 1.00 TA TATATATAATATA
6487 6492 2 3.0 2 100 0 12 100 0 0 0 0.00 AA AAAAAA
6516 6560 1 45.0 1 59 0 45 80 0 4 15 0.87 A AAATATATAAAAGATAAAAAAAATAAGAAAAATAAAAAAAATAAA
6977 6999 2 12.0 2 72 9 26 43 4 0 52 1.21 TA TATAATATTCTTATATATATATA
7090 7166 2 39.5 2 56 23 31 55 7 5 31 1.50 AT ATATATATTAAATAAGATAAAAAGTAAAATACTAAAAATATACCTACTATACTAGAGAATACTTATATAATATAATA
7808 7815 1 8.0 1 100 0 16 100 0 0 0 0.00 A AAAAAAAA
7889 7893 1 5.0 1 100 0 10 0 0 100 0 0.00 G GGGGG
```

1. *The summary table.*
   The information generated by the summary table above is discussed in more detail in Chapter 3. Briefly the summary table above provides the following information:

   - Indices of the detected TR relative to the start of the sequence.
   - The period size of the TR. This is the most common matching distance between corresponding characters in the alignment, and usually corresponds to the motif length of the consensus motif.
   - The number of repeats aligned with repeats of the consensus motif.
   - The length of the consensus motif (PTRE). (If the consensus motif is `TTC` then the length of the consensus motif is 3.) This may differ slightly from the period size.
   - The overall percentage of matches between adjacent repeats in the TR.
   - The alignment score. If the weight assigned to each matching character is equal to 2, and if there are x matching characters in the alignment then the alignment score will be $2x$. The alignment score refers to matches of the consensus sequence[2] and the relevant TR, detected in the genetic sequence.
   - Percentage of composition of the four nucleotides (adenine (`A`), cytosine (`C`), guanine (`G`) and thiamine (`T`)).
   - The measure of entropy[3] based on percent composition.

   In order to obtain more detailed information about a specific TR, the user can click on the applicable index range. This will result in an alignment explanation of the index range, as explained below.

2. *The alignment explanation*
   The alignment explanation includes the actual genetic sequence stretching throughout the TR (e.g. from indices 233 - 246 in the forthcoming example). It corresponds to and elaborates on the information presented in the above-described summary table that was created by Tandem Repeats Finder. Two alignment explanations that were generated during the running of `swam.txt` are included below.

```
Tandem Repeats Finder Program written by:
```

[2]The consensus sequence is a sequence of adjacent PTREs, repeated as many times as there are TREs in a string under consideration.

[3]The entropy estimation of a DNA sequence provides a measure of its complexity and randomness level [Vinga & Almeida (2004)].

```
                Gary Benson
       Department of Biomathematical Sciences
           Mount Sinai School of Medicine

Version 4.00

Sequence: swam

Parameters: 2 3 5 80 10 10 3

Pmatch=0.80,Pindel=0.10
tuple sizes 0,4,5,7
tuple distances 0, 29, 159, 500
Length: 8400
ACGTcount: A:0.29, C:0.21, G:0.23, T:0.27


Found at i:3683 original size:3 final size:3

<A NAME="3771--3795,2,12.5,2,12"></A>

    Indices: 3771--3795  Score: 13
    Period size: 2  Copynumber: 12.5  Consensus size: 2

       3761 AGCATGTTAC

                     ** *   *       *
       3771 TA TA TA GC AA TC TA -A AA TGA TA TA T
          1 TA TA TA TA TA TA TA TA T-A TA TA T

       3796 TTGCAGGTTT

Statistics
Matches: 13,  Mismatches: 8, Indels: 4
        0.52            0.32        0.16

Matches are distributed among these distances:
   1    1  0.08
   2   10  0.77
   3    2  0.15

ACGTcount: A:0.48, C:0.08, G:0.08, T:0.36

Consensus pattern (2 bp): TA

Found at i:6476 original size:2 final size:2

<A NAME="6471--6483,2,7.0,2,29"></A>

    Indices: 6471--6483  Score: 21
```

```
    Period size: 2  Copynumber: 7.0  Consensus size: 2

       6461 TTTTAATGAC


       6471 TA TA TA TA -A TA TA
          1 TA TA TA TA TA TA TA

       6484 GGGAAAAAAG

Statistics
Matches: 10,  Mismatches: 0, Indels: 2
        0.83            0.00        0.17

Matches are distributed among these distances:
   1    1  0.10
   2    9  0.90

ACGTcount: A:0.54, C:0.00, G:0.00, T:0.46

Consensus pattern (2 bp): TA
```

The consensus sequence is provided below the TR. The consensus sequence has been discussed in Chapter 3.

Additional information that accompanies the alignment explanation is as follows:

- The 10 base pairs, before and after a TR are shown by default. (As discussed in Chapter 3.)
- The symbol * appears above the pair of aligned lines to indicate a mismatch. In example 1 a `T` has been replaced by a `C`. Thus, a * appears above the `C` of the detected TR.
- The symbol - appears within the applicable sequence to indicate an insertion or a deletion. If a deletion occurred then - will appear within the *detected* TR. If an insertion occurred then - will appear within the *consensus sequence*. In example 2, a deletion has occurred. Thus a - appears within the detected TR, replacing the absent `T`.
- Statistics pertaining to matches, mismatches, insertions and deletions accompany the alignment explanation. These statistics have been discussed in Chapter 3.
- Distances between matching characters at corresponding positions are listed. These distances have been discussed in Chapter 3. Consider the data generated in terms of the distribution of matches in example 1:

```
Matches are distributed among these distances:
   3    9  1.00
```

Here 3 represents the distance. In the first example 9 matches occur - all the matching nucleotides have a matching distance of 3. Therefore the percentage of all matches occurring at distance 3 is 100% output as 1.00 by Tandem Repeats Finder.

Consider the output of the second alignment explanation. Here two distances are provided in the distance description as shown below:

```
Matches are distributed among these distances:
   1    1  0.10
   2    9  0.90
```

If there is referred to the indices where the base pairs occur, then it is possible to explain the number of nucleotides at a distance and the percentage of all matches as follows:

Distance 1 represents the two consecutive `A`'s occurring at index position 6478 and index position 6479. There is only 1 occurrence of matching nucleotides with a distance of 1. This is indicated by the second 1 in the top row of the distance distribution output. A detailed explanation of the distance distribution can be found in Chapter 3.

- The ACGT count reflects the percentage of each nucleotide.
- The motif termed the consensus pattern by Benson that constitutes 3 base pairs.
- The little additional output provided is not of importance for the current discussion.

6.3 STAR: Search for Tandem Approximate Repeats

As mentioned in Chapter 3, STAR runs on both a Linux system, as well as a Windows operating system. The results obtained from running STAR on Linux and Windows are similar. For the purposes of this dissertation STAR has been run on Linux. The input of STAR is discussed in detail in Chapter 3. It can be recalled, that at the command line prompt, STAR.linux should be entered. The following will appear on the screen:

```
-i SeqFile -m Motif | -M MotifFile
```

```
          [-na -po PositionOffset -help]
 SeqFile: file containing the sequence in Fasta, Genbank
          or Embl format
 Motif: the motif to search for is a string over
          alphabet [ACGT]
 MotifFile: a file with one motif per line, each motif
          is searched independently, this option
          excludes option -m
 -na: option without the output of alignments of
          tandem repeats;
 default is with alignments
 -po PositionOffset: set a position offset that is
          added to output positions;
```

Motifs introducing the longest tandem repeats detected by FireμSat were used, in order to determine the effectiveness of STAR during the trial run of `swam.txt`.

The motifs identified to run STAR with were as follows:

```
   A    T   TA  AAG  AAT  ACT  AGA  AGC  AGT  ATA
 ATC  ATG  CAC  CAG  CTA  GAT  GCT  GGT  GTA  GTG
 TAA  TAC  TAT  TCT  TGA  TGC  TGG  TTA  TTC  TTT
```

The output of STAR has been written to swam.a as follows:

```
ZONE    1 BEGIN_POS    6524 END_POS    6560 LG    37 GAIN   23
A   31 C    0 G    2 T    4 N    0 %AT  94 %GC   5 Biais GC 1.00
Phase 0
Consensus columns, counts of matches, substitutions, and deletions
Position     1
Nb_Match    31
Nb_Subst     0
Nb_Del       0
Insertion columns number: 1 and list of positions and counts
Position     1
Nb_Ins       6
         Match  Sub  Del  Ins
Totals      31    0    0    6
Percents  83.8  0.0  0.0 16.2
```

```
Nb_Motifs 31.00 Percent_of_exact_motifs 80.65 Consensus 1 a
Pat             aaaa-a-aaaaaaaa-aa-aaaaa-aaaaaaaa-aaa
Seq       6524  aaaaGaTaaaaaaaaTaaGaaaaaTaaaaaaaaTaaa
                    ^ ^         ^  ^     ^        ^

Nb_mutations 6
Mutations_list  5,s, 7,t, 16,t, 19,s, 25,t, 34,t,
FILE swam50.txt LG 8400 MOTIF_LG 1 MOTIF a NB_ZONES 1
```

For the above motifs entered FireμSat detected the following TRs:

motif	pos	len	TR	n,a,m,d,i
AAA	6523	36	AAAAGATAAAAAAAATAAGAAAAATAAAAAAAATAA	6,6,6,0,0
AAG	6539	21	AAGAAAAATAAAAAAAATAAA	1,6,6,0,0
AAG	6159	16	AAGTAGGAAGATGAAT	1,4,3,0,1
AAT	6516	44	AATATATAAAAGATAAAAAAAATAAGAAAAATAAAAAAAATAAA	4,11,8,2,1
AAT	6498	15	AATTATAATTATAAG	2,3,3,0,0
AAT	6563	15	AATAGATATAGTAAG	1,4,2,1,1
ACT	117	26	ACTATCTTCTACAAATTACTATCTAT	1,9,3,5,1
ACT	5988	16	ACTCTATATATATATT	1,6,1,5,0
AGA	1344	16	AGACGAGAAGCTGAGA	1,5,3,2,0
AGA	4699	15	AGAAGGATACGAATA	1,4,4,0,0
AGC	3156	14	AGCAGAAAGCCAGC	1,3,1,0,2
AGT	4598	18	AGTATAGCTGTAGCGTGT	1,6,3,3,0
ATA	6517	43	ATATATAAAAGATAAAAAAAATAAGAAAAATAAAAAAAATAAA	4,11,7,3,1
ATA	6990	16	ATATATATAATTACTA	1,5,1,3,1
ATA	6593	15	ATAATTTAATTTATA	1,5,2,3,0
ATA	7158	15	ATATAATAAGTAATA	3,2,0,1,1
ATC	1458	15	ATCATCAATCTCGTC	2,3,1,1,1
ATC	8039	15	ATCACAATCAGCATT	1,4,2,1,1
ATG	2204	14	ATGTGACGGTGATG	2,3,2,1,0
CAC	3682	14	CACTACCAGCGCCC	1,4,3,1,0
CAG	847	14	CAGCAAAGGCAGAG	1,4,1,2,1
CTA	7187	16	CTATATTATAGTAATA	1,5,3,2,0
GAT	6495	15	GATAATTATAATTAT	1,4,4,0,0
GCT	4507	15	GCTCCTGCAGCTGCG	2,3,3,0,0
GGT	7921	18	GGTGCTGGTGCTGGTAGT	3,3,3,0,0
GTA	6138	29	GTAGGTAGTATTATACTAGGTAAGTAGGA	2,7,3,1,3
GTA	6142	25	GTAGTATTATACTAGGTAAGTAGGA	2,6,3,1,2
GTA	5615	16	GTAGGTAGTTGTGATA	1,4,3,0,1
GTA	5884	16	GTAGATACTAATAGCA	1,5,3,2,0
GTA	4189	15	GTATGTATATTAATA	1,4,2,1,1
GTG	7922	20	GTGCTGGTGCTGGTAGTAGG	2,5,4,1,0
GTG	3549	16	GTGGATGGTCAGTGTG	1,4,1,1,2
TAA	6529	30	TAAAAAAAATAAGAAAAATAAAAAAAATAA	4,6,6,0,0
TAA	7113	17	TAAAATACTAAAAATAT	2,4,3,1,0
TAA	4877	16	TAATGATTAGTAATTA	1,4,3,0,1
TAA	6497	15	TAATTATAATTATAA	3,2,2,0,0
TAC	5879	19	TACTAGTAGATACTAATAG	1,5,4,0,1

```
TAC   7129 15  TACCTACTATACTAG                          1,4,2,1,1
TAT   6976 31  TATAATATTCTTATATATATATAATTACTAG          2,10,5,5,0
TAT   6981 26  TATTCTTATATATATATAATTACTAG              2,8,4,4,0
TAT   1144 16  TATTGATTACTACTGT                        1,4,3,0,1
TAT   6991 16  TATATATAATTACTAG                        1,5,3,2,0
TAT   4190 15  TATGTATATTAATAC                         1,4,2,1,1
TAT   5747 15  TATTTTGATTACTAA                         1,4,4,0,0
TAT   5996 15  TATATATTGTAATTT                         1,5,2,3,0
TCT   7235 17  TCTCTTTTCTCTTATCT                       1,6,2,4,0
TCT   7237 15  TCTTTTCTCTTATCT                         1,5,2,3,0
TGA   5845 14  TGATAATATGTTTA                          1,4,3,1,0
TGC    623 14  TGCAGCTGATCTGA                         1,4,3,1,0
TGG   7926 15  TGGTGCTGGTAGTAG                        2,3,3,0,0
TTA   6975 31  TTATAATATTCTTATATATATATAATTACTA         3,9,4,5,0
TTA    106 15  TTATTTTTCTCACTA                        1,4,4,0,0
TTA   6916 15  TTAATAATTATTTTA                        1,4,2,1,1
TTA    132 14  TTACTATCTATGTA                         1,3,1,0,2
TTC   7234 17  TTCTCTTTTCTCTTATC                      1,6,2,4,0
TTT   7232 18  TTTTCTCTTTTCTCTTAT                     1,5,5,0,0
```

This is contrary to STAR which detected only one TR in total.

The following information is output for the detected TR by STAR:

1. *General information.*
   The general information is written in the first two lines of the output:

   - `ZONE` (Line 1), All the TRs, listed in the same output file have the same consensus motif (identified PTRE). The zone number indicates that the detected TR is the first occurrence with the consensus motif "a", within the genetic sequence file `swam.txt`.
   - `BEGIN_POS` (Line 1), the start position of the TR is 6524.
   - `END_POS` (Line 1), the end position of the particular TR is 6560.
   - `LG` (Line 1), the length of the detected TR is 37.
   - `GAIN` (Line 1), the local compression gain is 23.
   - `A`, the number of occurrences of adenine is 31.
   - `C`, the number of occurrences of cytosine is 0.
   - `G`, the number of occurrences of guanine is 2.
   - `T`, the number of occurrences of thiamine is 4.
   - `N`, the number of occurrences of undetermined nucleotides were 0.
   - %`AT`, the percentage of weak hydrogen bonds, `AT` is 94.

- %`GC`, percentage of strong hydrogen bonds, `GC` is 5.
- `Biais` the `GC` bias occurrence is 1.00.
- `Phase`, indicates the pattern phase at the beginning of the TR, which is in this case 0. From Chapter 3 it can be recalled that the `Phase` contributes towards the positioning of the alignment [Delgrange & Rivals (2004a)].

2. *Consensus description*
   The consensus description is represented by a table, in which the columns represent the respective positions in the motif. (For more detail Chapter 3 can be consulted.)

   The entry in the first row (*Nb_Match*) at column 1, is 31. Note that there are neither deletions nor mismatches in the detected TR. The number of insertion columns detected within the detected TR, is 1 - there occurred according to the calculations of STAR 6 insertions within column 1 - the only column.

3. *The list of positions and counts*
   From Chapter 3 it can be recalled that the list of positions and counts consists of a table in which column entries indicate where insertions occurred in relation to the consensus motif for the TR under exploration. The number of insertion columns detected within the detected TR is 1 - there occurred according to the calculations of STAR 6 insertions within column 1.

4. *Distribution of matches, substitutions,deletions and insertions*
   The 4 columns of the table that represent the distribution of matches, substitutions, deletions and insertions, contain the headings $Match$, Sub, Del and Ins respectively. For more detail, Chapter 3 can be consulted. Regarding the consensus motif of the detected TR there occurs 31 matches and 6 insertions, thus 83.8% of the detected TR constitutes exact matches while 16.2% of the TR consists of insertions.

5. *Number of repeats and consensus pattern*
   An output line that contains the following information is provided:

   - *Nb_motifs*: the number of motif repeats that occurred in the detected TR. In the case of the above example, is 31.00 repeats.
   - *Percent_of_exact _motifs*: The ratio between the number of exact matches and the number of mutations is provided, which is shown to be 80.65%.
   - *Consensus*: This serves as a boolean flag that indicates whether a particular motif, entered as parameter, was detected or not. In this case $Consensus = 1$. The motif namely, a, under consideration is output alongside the consensus flag.

6. *Alignment*
   The detected TR is aligned against a concatenation of consensus motifs (concatenated repeats of the detected PTRE - a PTR). The alignment is computed by Wrap Around Dynamic Programming (Section 3.2). The output is written in four separate lines for each TR. There is a maximum of 60 characters in each line. The first output line is labelled as *Pat*. This line contains concatenated copies of the consensus motif. The second output line is labelled as *Seq*. *Seq* contains the actual detected TR. In the third line, each mutation is marked with a $\wedge$. Furthermore, irrespective of whether the input format of the genetic sequence was upper or lower case, upper case is used to indicate mutations in *Pat* and *Seq*, while all the other nucleotides are output in lowercase. The fourth line is empty.

7. *Mutations list*
   The mutation list consists of the list of mutations that occurred within a TR. The first line of output contains the number 6, the total number of mutations (*Nbmutations*) that occurred within the detected TR.

   It can be recalled from Chapter 3 that Delgrange & Rivals (2004a) developed single symbol codes to represent the entire spectrum of possible mutations. These codes are provided in Chapter 3 in Tables 3.2, 3.3 and 3.4. An explanation of these codes can also be found in Chapter 3. The output of the mutation list is:
   5, s;
   7, t;
   16, t;
   19, s;
   25, t; and
   34, t.

   The final line of the output of STAR repeats the name of the input file, swam50.txt; the length of the input file, 8400 base pairs; the motif length of the consensus motif namely, 1; the consensus motif itself, a and the number of detected TRs namely, 1.

6.4 FireμSat

6.4.1 FireμSat: Input

The Windows graphical users interface (GUI) of FireμSat has been discussed in Chapter 5 and is included in Figure 6.1. During the trial runs we only report on

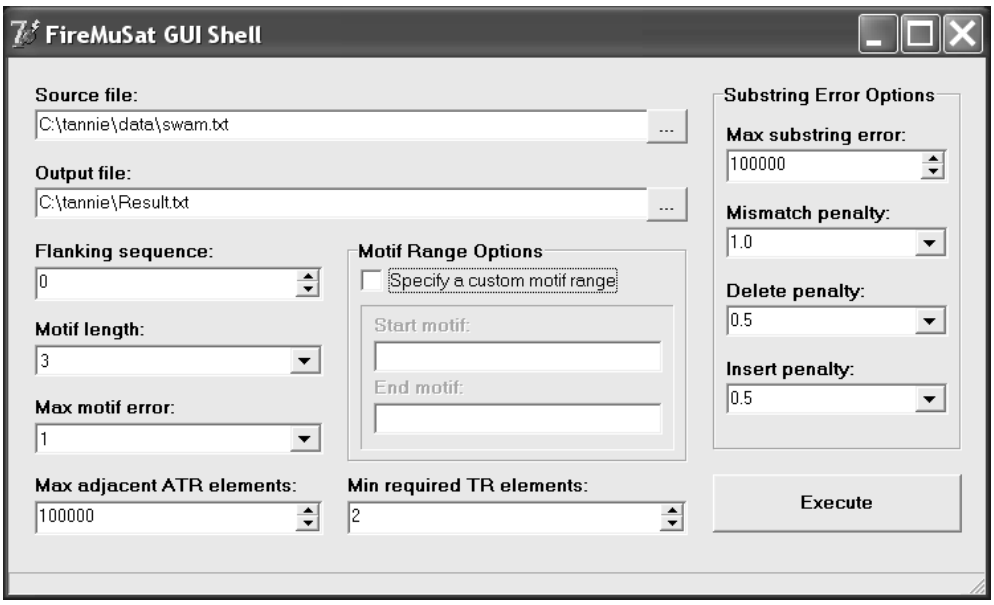


Figure 6.1: The GUI of the FireμSat software

FireμSat$_2$. Note, FireμSat$_1$ will generate data similar to FireμSat$_2$ while FireμSat$_3$ report each TR once.

If the reader is unclear regarding any of the terminology or input data of FireμSat then Chapter 5, Section 5.2 should be consulted. FireμSat has been run on `swam.txt` using the the following parameter set:

- `Source file:` The source file has been specified as `swam.txt`. Note the complete path to the input file has been specified as required.
- `Output file:` The output file has been specified as result.csv. Note the complete path to the input file has been specified as required. The reasons why the output file is specified as a .csv file and not as a .txt file have been discussed in Chapter 5, Section 5.2.
- `Flanking sequence:` For the trial run the flanking sequence has been set to 0, therefore no flanking sequence is output.
- `Motif length:` A motif length of 3 has been input for the trial run, thus the length of the PTREs that are being detected is 3.
- `Max motif error:` The maximum motif error for a PTRE of length 3 has been selected as 1, thus a motif error of 33.3% has been allowed during the trial run.

- `Max adjacent ATR elements`: To prevent a restriction on the number of ATR elements that occur next to each other, the largest possible value allowed by FireμSat has been selected namely 100000.
- `Maximum substring error`: The maximum substring error has not been calculated during the trial runs.
- `Mismatch penalty`: The mismatch penalty has been set to 0.
- `Deletion penalty`: The deletion penalty has been set to 0.
- `Insertion penalty`: The insertion penalty has been set to 0.

6.4.2 FireμSat: Output

Similar to the input of FireμSat the output of FireμSat and its accompanied terminology have been discussed in Chapter 5. FireμSat detected 2139 TRs of motif length 3 in `swam.txt`. The detected TRs are included on the CD that accompanies this dissertation. It is thus possible to conclude that FireμSat is more accurate than both Tandem Repeats Finder and FireμSat. An output example of one of the detected TRs is included.

```
;motif, TR, tr_start_pos, tr_length, n_ptre, n_atre, n_m, n_d, n_i
;>swam
TTA, TTATAATATTCTTATATATATATAATTACTA, 6975, 31, 3, 9, 4, 5, 0
```

The TR detected in the above case is introduced by the motif `TTA`. The start position of the detected TR is 6975 and the length of the detected TR is 31 base pairs. The value of `nptre` is equal to 3. Thus there is 3 exact copies of the PTRE or motif `TTA`. The total number of ATREs is 9. 4 of the detected ATRE's contain mismatches and 5 of the detected ATREs contain deletions.

6.5 A Tabular Comparison Between Software

Four tables have been composed to present the results, one containing the runtime results (Table 6.1) and three with the number of detected TRs for three datasets. The parameters are set as discussed.

Note, the Fusarium Graminearum genome, approximately 33 MB of data, is considered to be a relatively large file in the present problem domain. Since the processing speed of Tandem Repeats Finder and FireμSat$_2$ is fast, it is preferable to compare the execution times on a large set of data.

	TRF	STAR	FireμSat$_2$
Runtime	12 minutes 48 seconds	More than 3 hours	127 seconds

Table 6.1: Execution time comparison: results for Fusarium Graminearum.txt

	TRF	STAR	FireμSat
Number of TRs detected	16	1	2139

Table 6.2: Results for swam.txt

To gain an understanding of how effectively the respective packages identify all TRs in a given file and to answer the question if all possible TRs are identified or not there is reported on trial runs using the files swam.txt (8500 bytes) and Cylindrocladium.txt. Had Fusarium Graminearum.txt been used for the experiment of detecting all the TRs of motif length 3, an excessively large number of TRs would be detected during trial runs. Therefore, in the interest of both accuracy and simplicity it was decided to use the two smaller files to compare the generated data in a meaningful manner and to search Fusarium Graminearum.txt only for two motifs `ACG` and `TGA`.

The respective results are reported on in Table 6.2, Table 6.3 and Table 6.4. The number of TRs detected by FireμSat$_2$ is considerably larger than the number of TRs detected by both TRF and STAR. Note if the duplicate data that FireμSat$_2$ reports on is eliminated from the results of swam.txt then there will be 752 different TRs detected. Thus the number 2139 includes duplicate data reported on by FireμSat$_2$. So does the data in Table 6.4 also report duplicate data. If the duplicate data is eliminated then there will be reported on 2650 repeats of which 221 have a TR-length greater or equal to 15 base pairs. Tandem Repeats Finder and STAR detect a subset of the repeats detected by FireμSat$_2$.

Note that the data of Cylindrocladium Pauciramosum constitutes various short sequences in one file.

STAR does not currently provide the complete facilities to run data of this type. Therefore it is not practical to use STAR as a detection tool of TRs on Cylindrocladium Pauciramosum.

Motif	TRF	STAR	FireμSat (longer than 15bp)
ACG	243	16	3036
TGA	309	18	5894

Table 6.3: Number of TRs detected : Results for Fusarium Graminearum.txt (33MB)

	TRF	STAR	FireμSat
Number of TRs detected	209	NA	7952

Table 6.4: Results for Cylindrocladium Pauciramosum (WrightSEQ2.txt)

	FireμSat$_2$	IMEx
Runtime	127 seconds	1 hour 34 minutes

Table 6.5: Execution time comparison: results for Fusarium Graminearum.txt

6.5.1 Comparing FireμSat to IMEx

As mentioned in Chapter 1 the software package IMEx [Mudunuri & Nagarajaram (2007)] has been released in 2007. IMEx is also considered to be a prominent software package that detects microsatellites, allowing for mismatches, insertions and deletions, it is not discussed at the same level of detail as TRF and STAR as it has only been made available after most of this study had been completed. A comparison between the runtime of IMEx and FireμSat$_2$ as well as a comparison between the data generated by IMEx and the data generated by FireμSat$_2$ is included in Table 6.5 and Table 6.6 respectively.

Note, IMEx provides various parameters that can be set. For the purposes of the trial runs the parameters were set as loosely as possible. *Repeat size: tri*, *Minimum Repeat Number: 1*, *Imperfection Limit/repeat unit:3*, *% Imperfection in Repeat Tract:50%*. IMEx has been run from its web server. During the runtime trial the 33MB of data was uploaded to the IMEx server. Time subtracted for uploading was 30 seconds.

Table 6.5 shows that the runtime, FireμSat$_2$ was about 42 times faster than IMEx. For the data comparison, the file swam.txt was used. The input parameters used correspond exactly with those used during the trial runs of the execution time comparison.

The results are reported in Table 6.6. IMEx reports on 29% of the TRs that is detected by FireμSat$_2$. Note duplicate data reported by FireμSat$_2$ has been eliminated by a simple spreadsheet operation.

If the results of IMEx are compared to the results of FireμSat$_2$ then it may seem that both IMEx and FireμSat$_2$ divide some microsatellites in two or more — in some cases IMEx reports longer TRs than FireμSat$_2$ in others FireμSat$_2$ reports longer TRs than IMEx. In the case of FireμSat$_2$ reporting shorter or divided

	FireμSat$_2$	IMEx
Number of TRs detected	752	217

Table 6.6: Data comparison: results for swam.txt

	At index	IMEx reported	FireμSat$_2$ reported
	1281	1	2
	1457	1	3
	3911	1	5
*	5338	1	5
	6127	1	3
	7131	1	2
	7720	1	2
	7888	1	3

Table 6.7: Data comparison: divided microsatellites

TRs the division of some of the TRs is partially a consequence of the order of precedence during the identification of TREs. It can be recalled from Chapter 4 that if a final state is of multiple types, then the PTRE counter takes precedence, followed by the mismatch counter, followed by the deletions counter, followed by the insertion counter. By this is meant that if a state is encountered that is final for both PTREs and mismatches, then the PTRE counter is incremented rather than the mismatch counter. Similarly, mismatches are incremented rather than deletions, etc. The functionality to generate data of all meaningful combinations of precedence order is relatively easy to implement, and has been added to the commandline version of FireμSat$_2$, see Section 5.3.

Meaningful in this context implies either deletions are given precedence over mismatches or mismatches are given precedence over deletions (default). The remaining cases where FireμSat$_2$ reported shorter TRs than IMEx can be understood by taking into account that FireμSat does not to allow for more than one motif error if the motif length of the identified TR is equal to 3. IMEx does not follow a similar approach — if the data generated by IMEx is carefully analysed then it becomes apparent that IMEx allows for more than one motif error at times if the motif length is 3.

A further analysis of the data generated by FireμSat$_2$ and IMEx shows that FireμSat$_2$ divides microsatellites reported on by IMEx only in eight cases in two or more. The data can be found in Table 6.7. It is important to realize that none of the other TRs detected by IMEx has been divided or reported on more than once after duplicate data has been eliminated.

If the microsatellites that FireμSat$_2$ breaks up, compared to IMEx, are counted

*	FireμSat$_2$	IMEx
Number of TRs detected	736	217

Table 6.8: Data comparison: results for swam.txt

as one for each respective microsatellite as detected by IMEx then a comparison of the number of microsatellites detected can be seen in Table, 6.8:

Apart from the microsatellites listed in Table 6.8 IMEx detected longer microsatellites than FireμSat$_2$ in 16 cases. FireμSat$_2$ detected longer microsatellites than IMEx in 17 cases. It can be concluded that in spite of the fact that FireμSat$_2$ breaks certain microsatellites in two it does detect quite a number of microsatellites that are NOT detected by the available version of IMEx. IMEx does not detect any microsatellites that are not detected by FireμSat$_2$ at all. It should be emphasized that FireμSat$_2$ does provide its user with the ability to fine tune its input parameters. Therefore the output of redundant data can be prevented as discussed.

After the results of IMEx had been compared to those of FireμSat$_2$ it was decided to add a backward search and to the functionality of the commandline version of FireμSat$_2$ in order to augment results. The addition entailed adding a backward search to the initially only forward searching algorithm, from the introductory motif position, and augmenting the results accordingly. The results obtained after the backward search had been implemented are presented in Figure 6.3. Figure 6.2 shows the histogram of the comparative results between IMEx and FireμSat$_2$ without the backward search implementation. From Figure 6.2 and Figure 6.3 it can be concluded that the adjustment did not lead to a significant TR-length improvement.

6.5.2 Remarks

A few observations that seem relevant regarding the software comparisons is listed below:

- It is known that the current version of FireμSat_2 may report duplicate TRs. This happens when a TR is reported that has a motif which is a suffix of the motif of some other reported TR, and both TRs end at the same place. This means, effectively, that one TR is contained within another.

  Notwithstanding this potential error source, after checking, it was established that the 752 TRs reported by FireμSat$_2$ did not include any duplicate TRs of this nature.

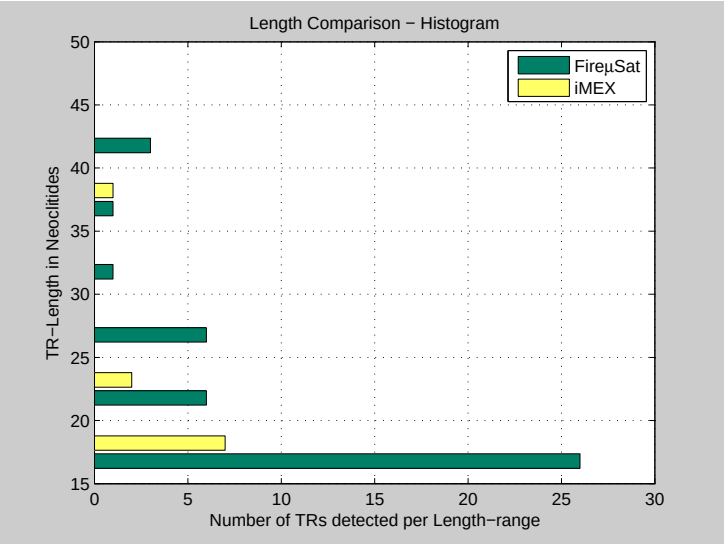


Figure 6.2: A histogram comparing the TR-length distributions of FireμSat$_2$ and IMEx respectively (for TRs longer than 15 base pairs) without backward searching.

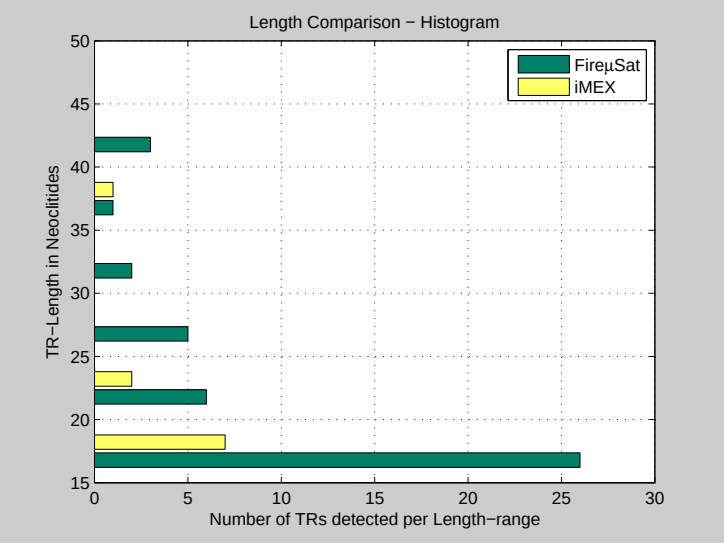


Figure 6.3: A histogram comparing the TR-length distributions of FireμSat$_2$ and IMEx respectively after the both-way search has been implemented.

- We have already remarked on the ambiguity in determining the most relaxed parameter settings for the TRF package. It seems relevant to note that the developers of IMEx report that they used the most relaxed set of input parameters for TRF namely: `Match=2`, `Mismatch = -7`, `Delta =-7` and `Min Score = 2`. They report on an experiment using these settings in

which IMEx detected 876 repeats for a set of data, while TRF only detected 50 repeats. Mudunuri & Nagarajaram (2007) reports on the details of this particular trial run.

Indeed, their claim initially seemed justified, because when these settings were used in a trial runs on our data, one repeat more TR was found than before, namely 17 in total instead of 16. However, upon closer inspection, it was found that two of these 17 TRs could be concatenated into a single TR that was not included in the TRs provided by the settings suggested by Mudunuri & Nagarajaram (2007), while this single TR was indeed one of the TRs identified by TRF run with our settings above.

- It should be noted that TRF was not developed to search for microsatellites *per se*. It was used here because of its prior use for benchmarking purposes [Mudunuri & Nagarajaram (2007), Delgrange & Rivals (2004b), and Wexler *et al.* (2005)].
- TRF, STAR and IMEx detected a subset of the repeats detected by FireμSat$_2$.

In the next section, Section 6.6 Tandem Repeats Finder, STAR and FireμSat will be evaluated against the criteria presented in Chapter 2.

6.6 Comparing the Tools

The criteria listed below have been compiled in Chapter 2. It is proposed that these criteria will contribute to the successful development of software tools for the detection of microsatellites. The list constitutes criteria proposed by Benson (1999), a criterium suggested by Delgrange & Rivals (2004b) and two of our own criteria.

The reader may consult Chapter 2 for more detail regarding the criteria listed below:

1. The avoidance of full scale alignment matrix computations in the case of alignment algorithms.
2. No *a priori* knowledge should be required pertaining to the pattern, pattern size or number of copies of the TR.
3. No restrictions should exist regarding the size of the repeats that can be detected.
4. Percentage differences between adjacent copies should be used and substitutions and indels should be treated separately.

5. A consensus pattern for the smallest repetitive unit in the TR should be determined.

6. The systematic detection of significant TRs in a way that is independent of the motif, should be made by an exact algorithm.

7. Flexibility: an algorithm that detects microsatellites should be flexible in terms of penalties awarded to indels and mismatches.

8. Useability: software to detect microsatellites should be useable specifically in terms of output. By this we mean that analytically, biologically and statistically relevant output should be provided to the user. Furthermore, we suggest a hierarchical output that will enable the user easily to obtain the most relevant data.

In the remainder of this section it will be argued that FireμSat complies to most of these criteria. The enumerated list below corresponds numerically to the criteria above and explains to which extend FireμSat satisfies each of these criteria. There will also briefly be referred to Tandem Repeats Finder and STAR.

1. FireμSat does not implement matrix alignment in order to detect TRs thus by implication full scale alignment matrix computations are avoided. Both Tandems Repeat Finder and STAR do comply to this criteria.

2. FireμSat does not require any *a priori* knowledge pertaining to the pattern or number of copies of the TR. At this stage FireμSat is developed to detect only microsatellites and can detect microsatellites of any length. Tandem Repeats Finder does not require any *a priori* knowledge pertaining to the pattern or number of copies of the TR. Tandem Repeats Finder detects microsatellites, minisatellites, as well as satellites. STAR needs the motif to be input of which TRs should be detected — similarly to FireμSat STAR only detects microsatellites.

3. FireμSat does not restrict the size of the repeats that can be detected in any manner, neither does Tandem Repeats Finder or STAR.

4. FireμSat enables the user to penalize substitutions, deletions and insertions separately during the calculation of the substring error by entering values for for `Mismatch penalty:`, `Deletion penalty:` and `Insertion penalty:` respectively. The percentage differences between adjacent copies are determined by the user who enters a value for the motif error - `Max motif error:`. Tandem Repeats Finder provides the possibility for the user to influence the penalization of alignment parameters - lower weights entered allow alignments with more mismatches, insertions and deletions, see Chapter 3, Section 3.1.1 for more detail. However, the calculation that determines whether

there should be reported on a detected TR or not also depends on other parameters (Chapter 3, Section 3.1.1 can be consulted in this regard) and is much more complicated than that of FireμSat. Consequently, the user cannot determine before hand exactly which types of TRs there will be reported on. STAR does not allow the user to penalize mismatches, insertions and deletions.

5. FireμSat does not determine a consensus pattern for the smallest repetitive unit in the TR. However, FireμSat determines a consensus pattern in the sense that for a detected TR a PTRE will always be output such that all TREs can be deduced from the PTRE within the restrictions determined by the user in terms of the motif error, the substring error and the maximum number of adjacent ATREs. Tandem Repeats Finder determines a so called consensus pattern. STAR functions regarding to a consensus pattern similarly to FireμSat.

6. The systematic detection of significant TRs in a way that is independent of the motif should be made by an exact algorithm. FireμSat detects microsatellites in a systematic manner independent of the motif and the user knows exactly the attributes of the detected TRs. If there exists a TR that complies to all the attributes specified by the user then the TR will be detected and will be output. It is possible to predetermine exactly which attributes detected TRs should have. The authors of STAR claim that their algorithm is exact in the sense that all TRs are reported on that yields to their compression criterium.

7. FireμSat is flexible in terms of the penalties awarded to indels (insertions and deletions) and mismatches. The user can predetermine the different penalties allocated to insertions, deletions and mismatches by entering values for `Mismatch penalty:`, `Deletion penalty:` and `Insertion penalty:` respectively. Tandem Repeats Finder allows the user to allocate weights to alignment parameters (`Match`, `Mismatch` and `Delta` determines whether a certain stretch of DNA should be reported on). Lower weights allows for more mismatches and indels. These parameters are used to determine the optimal alignment calculated during the run of the program. Together with the alignment parameters, the detection parameters `PM` and `PI`, determines whether a certain stretch of DNA should be reported on as a detected TR or not. The user cannot alter the values of the detection parameters, neither can the user predict the TRs he/she expects to be detected easily. The penalty weights for deletions and insertions are always equal. The reader is referred to Chapter 3 for more detail pertaining to the input values of Tandem Repeats Finder. It is not possible for the user to alter the penalties of either insertions, mismatches or deletion if they are using STAR. In this

regard Chapter 3 can be consulted.

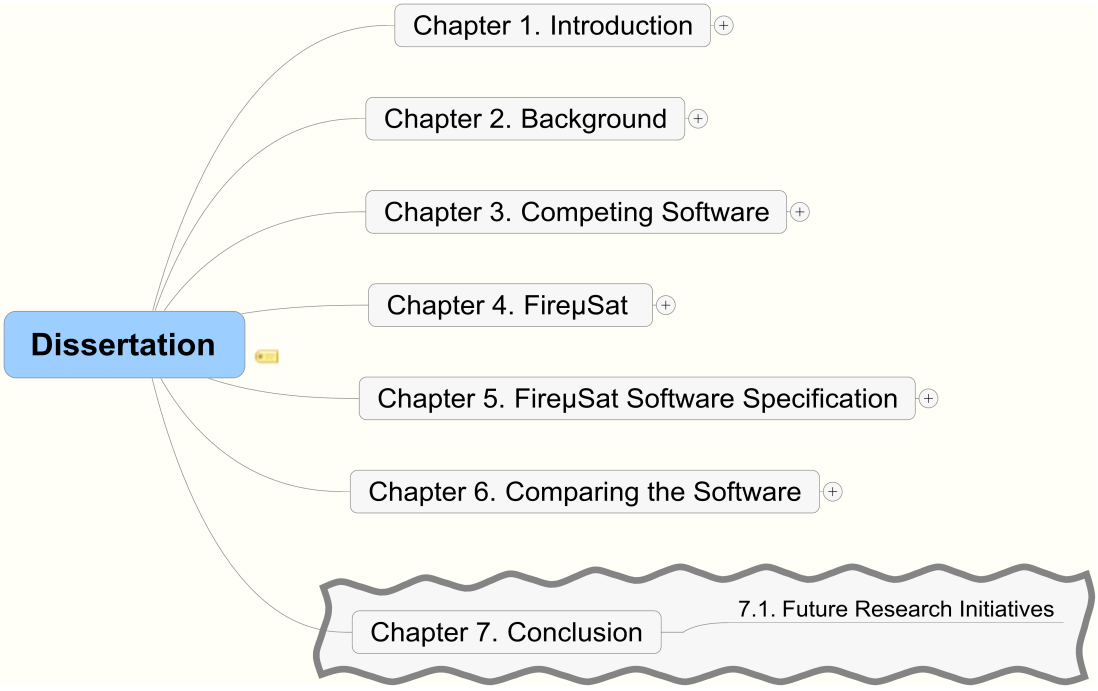
8. Useability: if the output of FireμSat is in .csv file format then the output can easily be arranged from the longest detected TR to the shortest detected TR by using any spreadsheet application. For the molecular biologist the longest TRs are those he/she prefers to investigate first. Furthermore, the calculation of FireμSat to determine whether or not a detected TR will be reported on is simple - no complicated mathematics skills are required to decide how penalties should be allocated or what the effect of any entered parameter will be. Both of these aspects contribute to the useability of FireμSat. It is also possible to import the .dat file output by Tandems Repeat Finder into a spreadsheet. However, importing the .dat file may require some changes and not all the relevant fields will be available in the report. Fields available are as follows: `Indices`, `Period Size`, `Copy Number`, `Consensus Size`, `Percent Indels`, `Percent Matches`, `A`, `C`, `G`, `T` and `Entropy`. The alignment parameters that can be entered by the user of Tandems Repeats Finder has an influence on the TRs that are reported. However, it is not simple to predetermine the exact type of repeats that will be reported on. STAR does not allow the user to modify penalties allocated to mismatches, insertions and deletions.

6.7 Conclusion

It has been argued that FireμSat complies to the relevant, in terms of microsatellites, criteria compiled in Chapter 2. It is clear that FireμSat makes provision for the accurate detection in a simpler, but more accurate manner than both STAR and Tandem Repeats Finder. Simpler in this context refers to the ease with which a user can allocate penalties and predetermine the exact nature of the TRs that will be detected. From the comparison of output during the trial runs discussed in this chapter it is clear that FireμSat detects more TRs than both STAR and Tandem Repeats Finder. Chapter 7 will conclude this dissertation.

Chapter 7

Conclusion



To conclude this dissertation consider the research hypothesis that has been introduced in Chapter 1:
FAs can effectively detect microsatellites on DNA.

Three variants of FireμSat have been introduced. The introduced algorithms rely on the implementation of FAs. All three variants FireμSat$_1$, FireμSat$_2$ and FireμSat$_3$ have been implemented.

Hereafter reference to FireμSat will refer to all the these packages. A comparison has been made between FireμSat$_2$ and two identified competitive software packages TRF and STAR. The runtime of FireμSat$_2$ is faster than that of TRF and STAR. TRF and STAR detect a subset of the microsatellites detected by FireμSat. Additionally the data generated by FireμSat has also been compared to data generated by IMEx in Chapter 6. (IMEx was released after the major part of this study had been completed.)

It was found that FireμSat detects all the microsatellites detected by IMEx. However, as a consequence of certain design decisions, FireμSat divides some of the microsatellites detected by IMEx into more than one. Recall from Chapter 6 that FireμSat detects microsatellites that are not detected by IMEx. The runtime of FireμSat is faster than that of IMEx. The FireμSat software complies with the criteria to which software should adhere in searching effectively for microsatellites.

These criteria were introduced in Chapter 2. During the design of FireμSat special attention has been paid to the development of flexible software. Therefore the implementation of FAs to search for microsatellites in DNA resulted in software that allows the fine tuning of searches for microsatellites by manipulating input parameters.

From the above it can be concluded that FAs can be implemented to search for microsatellites in DNA effectively.

7.1 Future Research Initiatives

The following future research initiatives were identified:

- From the reflective research done, it is clear that there is a lack of clarity about the specifications to which software packages should comply to search effectively for microsatellites. In Chapter 2 criteria to which software packages should comply have been proposed. The question which arises is: *"How do research domains differ regarding these criteria ?"* This research has been specifically instigated by the needs of molecular biologists. Whether these criteria can be used effectively in other domains in which researchers are also searching for microsatellites, remains to be investigated.

- The implementation of FAs to search for microsatellites in DNA resulted in software that allows the fine tuning of searches for microsatellites by manipulating input parameters. More investigation should still be conducted to determine how generated data differs for different values of input parameters. It is relatively easy to change the priorities for the identification of

mutations. An analysis of data generated by FireμSat as a result of such priority changes should be conducted. Parameter settings of FireμSat to detect data of relevance in other knowledge domains should also be determined. Furthermore FireμSat should be compared to more software packages.

- FireμSat$_1$ has the potential to be extended to search for minisatellites too. Therefore, the requirements for software searching for minisatellites should also be investigated. Challenges that will have to be dealt with if one extends FireμSat$_1$ to search for minisatellites, include an investigation of how FAs can best be stored in memory. Reduction algorithms to minimize the number of states generated, should be investigated. Note that the classical FA minimization algorithms cannot be directly applied, since the counting semantics assigned to certain states should not be obscured by state merges. Techniques that indicate how run time can be optimized, should also be considered. Furthermore it should be investigated how to merge the theoretical underpinnings of FireμSat$_1$ and FireμSat$_3$.

- Note that further research should be conducted into the extent to which software packages that search for microsatellites comply with HCI (Human Computer Interaction) principles. If software packages do not comply to HCI principles then two questions should be addressed:

  1. In what sense do these packages not comply with HCI principles?
  2. What exactly will software packages that do comply to HCI principles and search for microsatellites be like?

  It was not a primary objective of this research to fully explore the use of HCI principles in the present context. Nevertheless, data generated by this research appears to be useable, and the users do have the ability to fine-tune their searches by setting parameters in a flexible manner.

Glossary, Abbreviations, Acronyms

Acronyms and Abbreviations

A The symbol used to represent the base pair Adenine.

AFLP Is an abbreviation for amplified fragment length polymorphism (the occurrence of different forms of a gene in members of the same specie).

ATR Approximate tandem repeat.

BCCL Bioinformatics Computational Core Laboratories.

BISTIC The Biomedical Information Science and Technology Initiative Consortium of the National Institution of Health of the United States Of America.

BLAST Basic Local Alignment Search Tool.

C The symbol used to represent the base pair Cytosine.

DDBJ DNA DataBank of Japan.

DFA Deterministic finite automaton.

DNA Deoxyribonucleic acid.

EMBL European Molecular Biology Laboratory.

EMBOSS The European Molecular Open Source Software Suite.

Entrez The widely used Entrez Search System, which allows for the retrieval of a wide range of molecular biology data and bibliographic citations.

FA Finite automaton.

FASTA The FASTA format defines the file format used to store and exchange information between genetic databases. A sequence in FASTA format begins with a single line description, that is followed by lines of sequence data. The description line is distinguished from the sequence data by means of a greater-than symbol, ">, in the first column of the first row. It gives a name and/or a unique identifier to the sequence. The description line may also give other relevant information.

G The symbol used to represent the base pair Guanine.

GenBank One of the largest public sequence databases.

GSX Mouse Gene Expression Database

GUI Graphical User Interface

indel Insertion or deletion.

Me Mealy machine.

MGDB Mouse Genome Database.

Mo Moore machine.

NCBI National Center for Biotechnology Information.

NDB Nucleic Acid Database.

PCR Polymerase chain reaction

PTR Perfect tandem repeat.

STAR Search for Tandem Approximate Repeats — a software package.

T The symbol used to represent the base pair Thymine.

TR Tandem repeat.

TRF Tandem Repeats Finder — a software package.

Definitions and Descriptions

Adenine A nucleic acid, base pair or nucleotide. Also see Cytosine, Guanine and Thymine.

approximate tandem repeat A genomic sequence whose introductory substring (or motif) is followed by one or more substrings, of which at least one need not necessarily be an *exact* copy of the motif.

Bioinformatics The application of computational techniques to understand and organize the information associated with biological macromolecules.

Cytosine A nucleic acid, base pair or nucleotide. Also see Adenine, Guanine and Thymine.

DNA polymerase An enzyme which catalyzes the addition of a nucleotide to a nucleic acid molecule.

entropy The entropy estimation of a DNA sequence provides a measure of its complexity and randomness level based on percent composition.

Finite automaton An FA is defined as: a finite set of states $Q = \{q_0, q_1, q_2...q_n\}$, of which q_0 is designated to be the start state; a subset of Q called the final states (F); a finite alphabet $\Sigma = \{x_1, x_2, x_3...x_s\}$; and a transition function $\delta : Q \times \Sigma \rightarrow Q$.

Guanine A nucleic acid, base pair or nucleotide. Also see Adenine, Cytosine and Thymine.

interspersed repeats Interspersed repeats are repeated DNA sequences located at dispersed regions in a genome.

low complexity DNA Low complexity DNA sequences are sequences that contain biological information that is considered to be biologically less relevant.

Mealy machine A six tuple $Me = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where Q, Σ, δ and q_0 are defined similarly to the formal definition of the FA. Δ is the output alphabet and λ maps $Q \times \Sigma$ to Δ , thus $\lambda(q, a)$ gives the output associated with the transition from state q on input a. The output of Me in response to input $a_1, a_2, \cdots, a_n$ is $\lambda(q_0, a_1), \lambda(q_1, a_2), \cdots, \lambda(q_{n-1}, a_n)$, where $q_0, q_1, \cdots, q_n$ are states such that $\delta(q_{i-1}, a_i) = q_i$ for $1 \leq i \leq n$. If the input sequence is of length n, then the output sequence is also of length n.

Moore machine A six tuple: $Mo = \{Q, \Sigma, \Delta, \delta, \lambda, q_0\}$ where: Q, Σ, δ, and q_0 are the same as in the FA. Δ is the output alphabet and, λ is a mapping from each state in Q to Δ, representing the output associated with each state. The output of Mo in response to input $a_1, a_2, \cdots, a_n, n \geq 0$, is $\lambda(q_0), \lambda(q_1) \cdots \lambda(q_n)$, where $q_0, q_1, \cdots, q_n$ represent the sequence of states such that $\delta(q_{i-1}, a_i) = q_i$ for $1 \leq i \leq n$. Notice that the Moore machine will give output of length $n + 1$ if n is the length of the input sequence

optimal alignment The optimal alignment of two sequences implies the determination of an alignment that will obtain the highest score (the score is

calculated by a function as specified by the developer) from aligning two given sequences in all possible manners can be found.

perfect tandem repeat A perfect tandem repeat (PTR) is a string of nucleotides in a genomic sequence whose initial substring (of some arbitrary length), is followed by one or more exact copies of that substring.

polymer molecule Polymer molecules are large molecules consisting of repeated chemical units joined together.

polymerase chain reaction A molecular biological technique for amplifying or copying a selected region of a DNA molecule, so that its sequence is multiplied many times in a laboratory.

primer A primer is an oligonucleotide (a short, single stranded DNA molecule synthesized chemically under automated conditions generally 15 - 50 nucleotides in length) which is complementary to a specific region within a DNA or a RNA molecule. Primers are used to initiate synthesis of a new strand of complementary DNA at that specific site, in a reaction or series of reactions catalyzed by a "DNA polymerase.

regular expressions Languages associated with regular expressions are regular languages. Regular languages are type 3 languages of the Chomsky hierarchy of grammars and are accepted by FAs.

sequencing Sequencing is the determination of the order of nucleotides in a DNA or RNA molecule or the order of amino acids in a protein.

short tandem repeats See microsatellites.

tandem repeat Is a string of nucleotides that is characterized by a certain motif which introduces the string, followed by at least one "copy of the motif.

Thymine A nucleic acid, base pair or nucleotide. Also see Adenine, Cytosine and Guanine.

Computational Biology The development and application of analytical and theoretical methods, mathematical modeling and computational simulation techniques to the study of biological, social and behavioural systems.

Bibliography

Abajian, C. (2003). Sputnik. Online: http://espressosoftware.com/pages/sputnik.jsp.

Altschul, S., Gish, W., Miller, W., Myers, E. W. & Lipman, D. (1990). A basic local alignment search tool. *Journal of Molecular Biology* **215**, 403–410.

Answers.com (2003). Peptide (in Biology): definition from The Columbia Electronic Encyclopedia. Online: http://www.answers.com/topic/peptide?method=5&linktext=peptide#copyright.

Benson, G. (1995). A space efficient algorithm for finding the best non-overlapping alignment score. *Theoretical Computer Science* **145**.

Benson, G. (1999). Tandem Repeats Finder. *Nucleic Acids Research* **27**(2), 573 – 580.

Benson, G. (2003a). How does Tandem Repeats Finder work? Online: http://tandem.bu.edu/trf/trfdesc.html.

Benson, G. (2003b). Tandem Repeats Finder: Definitions: FASTA Format. Online: http://tandem.bu.edu/trf/trf.definitions.html.

Benson, G. (2003c). Tandem Repeats Finder:Unix Version help: using Tandem Repeats Finder for Unix. Online: file://D:/Tandem%20Repeats%20Finder%20%20Unix%20Version%20Help.htm.

Benson, G. (2005a). Gary Bensonś Homepage. Online http //tandem.bu.edu/benson.html.

Benson, G. (2005b). Tandem Repeats Finder Download Page. Online: http://tandem.bu.edu/trf/ trf.download.html.

Bergeron, B. P. (2003). *Bioinformatics Computing*. Prentice-Hall/ Professional Technical Reference: Upper Saddle River, NJ.

Bio-Synthesis, Inc (2007). Bioinformatics glossary. Online: http://falcon.roswellpark.org/labweb/glossary.html.

Bioinformatics Computational Core Laboratories (2005). BCCL software. Online: http://www.vcu.edu/csbc/bccl/resources-software.htm.

Biology Online (2005). Chromosome in th Biology Online Dictionary. Online: www.biology-online.org.

Broad Institute (2009). Broad Institute of MIT and Harvard [homepage]. Online: http://www.broadinstitute.org.

Brown, S. (2004). Computers and the Human Genome Project: Smith-Waterman algorithm. Online: http://cse.stanford.edu/class/sophomore-college/projects-00/computers-and-the-hgp/smith_waterman.html.

Brown, S. M. (1995). Searching databases for sequences similar to a sequence of interest: H. Smith-Waterman searching. Online: http://www.med.nyu.edu/rcr/rcr/course/sim-sw.html.

Camp, N., Cofer, H. & Gomperts, R. (1998). High-Throughput. Online: http://www.sgi.com/ industries/-sciences/chembio/resources/papers/HTBlast/HT_Whitepaper.html.

Castelo, A. T., Martins, W. & Gao, G. R. (2002). TROLL: Tandem Repeat Occurrence Locator. *Bioinformatics Applications Note* **18**(4), 634–636.

Chao, K. M., Hardison, R. C. & Miller, W. (1993). Locating well-conserved regions within a pairwise alignment. *Computer applications in the biosciences (CABIOS)* **9**, 169–176.

Clamp, M. (2009). Jalview: a Java Multiple Alignment Editor. Online: http://www.jalview.org.

Cohen, D. I. A. (1997). *Introduction to Computer Theory*. Wiley: New York, 2nd ed.

Cooper, S. (2008). Translation and open reading frames. Online: http://bioweb.uwlax.edu/GenWeb/Molecular/Seq_Anal/Translation/translation.html.

Coward, E. & Drablos, F. (1998). Detecting periodic patterns in Biological sequences . *Bioinformatics* **14**, 498–507.

Cowen, L. J. (2002). COMP 150BIO: Computational Biology: Lecture 1: Sequence alignment (Part I). Online: http://www.eecs.tufts.edu/ cowen/biotalk/02lecture1.ps.

Dahl, O.-J., Dijkstra, E. W. & Hoare, C. A. R. (1972). *Structured Programming*, vol. 8. Academic Press.

De Ridder, C., Kourie, D. G. & Watson, B. W. (2006a). FireμSat: an algorithm to detect microsatellites in DNA. *Proceedings of the Prague Stringology Conference* .

De Ridder, C., Kourie, D. G. & Watson, B. W. (2006b). Meeting the challenge of detecting microsatellites in DNA. *South African Institute of Computer Scietists and Information Technologists:Service-oriented Software and Systems* .

Delgrange, O. (2005). Curriculum vitae: 21 Octobre 2005: Oliver Delgrange. Online: http://staff.umh.ac.be/Delgrange.Olivier/home.html.

Delgrange, O. & Rivals, E. (2004a). Appendix to the article: STAR: an algorithm to search for tandem approximate repeats. Online: http://atgc.lirmm.fr/STAR/downloads/STAR-Appendix.pdf.

Delgrange, O. & Rivals, E. (2004b). STAR: an algorithm to search for tandem approximate repeats. *Bioinformatics* **20**(16), 2812–2820.

Easterbrook, S., Singer, J., Storey, M.-A. & Damian, D. (2007). Selecting empirical methods for software engineering research. *Guide to Advanced Empirical Software Engineering* .

Farlex Inc. (2005). FASTA format in The Free Dictionary. Online: http://encyclopedia.thefreedictionary.com /FASTA format.

Gish, W. R. (2009). AB BLAST. Online: http://blast.advbiocomp.com/doc/README.html.

Glas, R. L., Ramesh, V. & Vessey, I. (2004). An analysis of research in Computing disciplines. *Communications of the ACM* **47**(6).

Google (2005). Definitions of Entrez on the Web. Online: http://www.google.co.za/search?hl=en&lr=&oi=defmore&q=define:Entrez.

Goubil-Gambrell, P. (1991). What do Practitioners need to know about Research Methodology? *Directions in Technical communications research* .

Grunwald, P., Myung, I.-J. & Pitt, M. (2001). NIPS 2001 Workshop: Minimum description length: developments in theory and new applications. Online: http;//quantrm2.psy.ohio-state.edu/injae/workshop.htm.

Guan, X. & Uberbacher, E. (1996). A fast look-up algorithm for detecting repetitive DNA sequences. *In: Proceedings of the Pacific Symposium on Biocomputing 1996* .

Hackett, B. & Gross, R. B. (2007). Providing Quality Molecular Biology Software Since 1984. Online: http://www.textco.com.

Herr, C. M. (2008). Chapter 2: Reproduction is the manifestation of heredity. Online: http://www.biology.ewu.edu/aHerr/Genetics/Bio310/Pages/ch2pages/genchap2.html.

Herrmannsfeldt, G. (1998). A highly parallel finite state automaton processor for Biological pattern matching. *In: Proceedings of the Prague Stringology Club Workshop 1998* .

Hopcroft, J. E. & Ullman, J. D. (1979). *Introduction to automata theory languages and computation.* Addison-Wesley publishing company.

Hyde, B. (2000). Data structures and search trees: class notes for Bio-Informatics. Online: www.msci.memphis.edu/ giri/compbio/f00/BryanHyde/Bryan.htm.

Johns, U. (2005). The GNU Project. Online: http://www.gnu.org/.

Kahn, C. E. (2005). BioInformatics Glossary: Quantitative PCR. Online: http://big.mcw.edu.

Kannan, S. K. & Myers, E. W. (1996). An algorithm for locating nonoverlapping regions of maximum alignment score. *SIAM Journal on Computing* **25**(3), 648–662.

Karlin, S., Morris, M., Ghandour, G. & Leung, M. Y. (1988). Efficient algorithms for molecular sequence analysis. vol. 85.

Kim, N.-S., Park, N.-I., Kim, S.-H., Kim, S.-T., Han, S.-S. & Kang, K.-Y. (2000). Isolation of TC/AG repeat microsatellite sequences for fingerprinting rice blast fungus and their possible horizontal transfer to plant species. *Molecules and Cells* **10**(2), 127–134.

Kolpakov, R. & Kucherov, G. (1999). Finding maximal repetitions in a word in linear time. *In: 40th FOCS. IEEE Computer Society Press* .

Kolpakov, R. & Kucherov, G. (2001). Finding approximate repetitions under Hamming distance. *In: ESA: Annual European Symposium on Algorithms, Lecture Notes in Computer Science* **2161**, 170–181.

Kourie, D. G. (2009). Formal Aspects of Computing.

Kressel, K. (1997). Practice-Relevant Research in Mediation: Toward a Reflective Research Paradigm. *Negotiation Journal* **13**(2), 143–160.

Krishan, A. & Tang, F. (2004). Exhaustive whole-genome tandem repeats search. *Bioinformatics* **20**(16), 2702–2710.

Kurtz, S., Choudhuri, J., Ohlebush, E., Schleiermacher, C., Stoye, J. & Giegerich, R. (2001). REPuter: the manifold applications of repeat analysis on a genomic scale. *Nucleic Acid Research* **29**.

Kurtz, S. & Schleiermacher, C. (1999). REPuter: fast computation of maximal repeats in complete genomes. *Bioinformatics Application Note* **15**(5), 426–427.

Landau, G. M. & Schmidt, J. P. (1993). An algorithm for approximate tandem repeats. *In: Proceedings of the 4th Combinatorial Pattern Matching Conference, Lecture Notes in Computer Science 648* .

Landau, G. M., Schmidt, J. P. & Sokol, D. (2001). An algorithm for approximate tandem repeats. *Journal of Computational Biology* **8**(1), 1–18.

Lee, F. (1996). *Interspersed repeats and Tandem Repeats in the Molecular Biology Web Book*, chap. 3.

Lefebvre, A., Lecroq, T., Dauchel, H. & Alexandre, J. (2003). FORRepeats: detects repeats on entire chromosomes and between genomes. *Bioinformatics* **19**(3), 319–326.

Lefers, M. (2004). Sequencing: Definition. Online: http://www.biochem.northwestern.edu/holmgren/Glossary.

Lemon, S. M. & Barbour, A. G. (1993). A glossary of terms commonly used in molecular biology. Online: http://www.med.unc.edu/wrkunits/3ctrpgm/pmbb/mbt/GLOS.htm.

Luscombe, N. M., Greenbaum, D. & Gerstein, M. (2001). What is Bioinformatics? A proposed definition and overview of the field. *Methods of Information in Medicine* **40**.

Main, M. & Lorentz, R. (1984). An O (n log n) algorithm for finding all repetitions in a string . *Journal of Algorithms* **5**, 422–432.

Milosavljevic, A. & Jurka, J. (1993). Discovering simple DNA sequences by the algorithmic significance method. *Computer Applications in Biosciences* **9**(4), 407–411.

Mudunuri, S. B. & Nagarajaram, H. A. (2007). IMEx: Imperfect microsatellite extractor. *Bioinformatics* **23**(10), 1181–1187.

Myers, G. & Sagot, M.-F. (1998). Identifying satellites and periodic repetitions in Biological sequences. *Journal of Computational Biology* **5**(3), 539–554.

National Center for Biotechnology Information (2005). National Center for Biotechnology Information homepage. Online: http://www.ncbi.nlm.nih.gov/.

National Institute of Health of the United States of America. BISTIC Definition Committee (2000). NIH working definition of Bioinformatics and Computational Biology. Online: http://www.bisti.nih.gov/CompBioDef.pdf.

Paces, J. (2001). Bioinformatics: Tools for analysis of Biological sequences. *In: Proceedings of the Prague Stringology Conference 2001* , 50–58.

Pestronk, A. (2005). DNA repeat sequences and diseases. Online: http://www.neuro.wustl.edu/neuromuscular /mother/dnarep.htm.

Rice, P. & Bleasby, A. (2009). EMBOSS: The Application. Online: http://emboss.sourceforge.net/.

Rivals, E., Delahaye, J. P., Delgrange, O. & Dauchet, M. (1995). A first step toward chromosome analysis by compression algorithms.

Rivals, E., Delgrange, O., Delahaye, J.-P., Dauchet, M., Delorme, M.-O., Henaut, A. & Ollivier, E. (1997). Detection of significant patterns by compression algorithms: The case of approximate tandem repeats in DNA sequences. *CABIOS* **13**, 131–136.

Safran, M., Solomon, I., Shmueli, O., Lapidot, M., Shen-Orr, S., Adato, A., Ben-Dor, U., Esterman, N., Rosen, N., Peter, I., Olender, T., Chalifa-Caspi, V. & Lancet, D. (2002). GeneCardsTM 2002: Towards a complete, object-oriented, human gene compendium. *Bioinformatics Applications Note* **18**(11), 1542–1543.

Sagot, M. F. & Myers, E. W. (1998). Identifying satellites in nucleic acid sequences. *In: Proceedings of the Second Annual International Conference on Computational Molecular Biology (RECOMB-98), edited by S. Istrail, P. Pevzner and M. Waterman. ACM Press* , 234–242.

Schmidt, J. P. (1998). All highest scoring paths in weighted grid graphs and its application to finding all approximate repeats in strings . *SIAM Journal on Computing* **27**, 972–992.

Smit, A. F. A., Hubley, R. & Green, P. (2003). RepeatMasker documentation. Online: http://www.repeatmasker.org/webrepeatmaskerhelp.html.

Stoye, J. & Gusfield, D. (2002). Simple and flexible detection of contiguous repeats using a suffix tree. *Theoretical Computer Science* **27**, 843–856.

Strachan, T. & Read, A. P. (2004). *Human Molecular Genetics*, vol. 3. Garland Science, 3 ed.

The Marine Biological Laboratory (2003). File formats. Online: http://workshop.molecularevolution.org.

The Marine Biological Laboratory. Workshop on Molecular Evolution (2004). FASTA DNA (aligned). Online: http://workshop.molecularevolution.org.

Theologis, A. (2001). Goodbye to one by one' genetics. *Genome Biology* **2**(4), 1–9.

Thurston, M. I. & Field, D. (2005). Msatfinder: Detection and characterisation of microsatellites. Online: http://www.genomics.ceh.ac.uk/ milo/msatfinder/.

Tran, N., Bharaj, B. S., Diamandis, E. P., Smith, M., Li, B. D. L. & Yu, H. (2004). Short tandem repeat polymorphism and cancer risk: influence of laboratory analysis of epidemiologic findings. *Cancer Epidemiology Biomarkers and Prevention* **13**, 2133–2144.

Trivedi, B. P. (2000). Sequencing the genome. Online: http://www.genomenewsnetwork.org/ articles/06_00/ sequence_primer.shtml.

Trochim, M. K. (2006). Deduction and Induction. Online: http://www.socialresearchmethods.net.

Van den Bergh, I. (2006). *Finding microsatellites in whole genomes.* Master's thesis, Technische Universiteit Eindhoven.

Van der Nest, M. A., Steenkamp, E. T., Wingfield, B. D. & Wingfield, M. J. (2000). Development of simple sequence repeat (SSR) markers in Eucalyptus from amplified inter-simple sequence repeats (ISSR). *Plant Breeding* **119**, 433– 436.

Vide, C. M., Mitrana, V. & Paun, G. (2003). *Grammars and Automata for String Processing.* CRC Press.

Vinga, S. & Almeida, J. S. (2004). Rnyi continuous entropy of DNA sequences. *Journal of Theoretical Biology* **231**(3), 337–388.

Volfovsky, N., Haas, B. J. & Salzberg, S. L. (2001). A clustering method for repeat analysis in DNA sequences. *Genome Biology* **2**(8).

Ware, D., McCouch, S., Buckler, E. & Jaiswal, P. (2005). Gramene Glossary. Online: http://www.gramene.org/documentation/glossary/.

Watson, B. W. (1994). The design and implementation of the FIRE Engine: A C++ toolkit for FInite automata and regular expressions. Online: http://alexandra.tue.nl/extra1/wskrap/public html/9411065.pdf.

Watson, B. W. (1995). Taxonomies and Toolkits of Regular Language Algorithms. Online: http://www.fastar.org/publications/PhD_Watson.pdf.

Watson, B. W. (1996). Implementing and using finite automata toolkits. *Nat. Lang. Eng.* **2**(4), 295–302.

Watson, B. W. (2001). A Taxonomy of Algorithms for Constructing Minimal Acyclic Deterministic Finite Automata **2214/2001**, 174–182.

Watson, B. W. (2002). A Fast and Simple Algorithm for Constructing Minimal Acyclic Deterministic Finite Automata. *j-jucs* **8**(2), 363–367.

Webopedia (2004). Perl: The Webopedia Computer Dictionary. Online: http://www.webopedia.com/ TERM/P/Perl.html.

Wexler, Y., Yakhini, Z., Kashi, Y. & Geiger, D. (2005). Finding approximate tandem repeats in genomic sequences. *Journal of Computational Biology* **12**(7), 928–924.

Wright, L. P., Wingfield, B. D., Crous, P. W. & Wingfield, M. J. (2007). Isolation and characterization of microsatellite loci in Cylindrocladium pauciramosum. *Molecular Ecology Notes* **7**(2), 343–345.

Zane, L., Bargelloni, L. & Patarnello, T. (2002). Strategies for microsatellite isolation: A review. *MOL ECOL* **11**, 1–16.