UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

University of Pretoria

Submitted in fulfilment of the requirements for the degree
Master of Science (Computer Science)
in the Faculty of Engineering, Built Environment and Information Technology,
University of Pretoria,
Pretoria,
South Africa

An investigation into the consistency and usability of selected minisatellite detecting software packages

by

Koos Themba Masombuka
97129102

Supervisor: Prof DG Kourie

October, 2013

To Nana my wife & Sn'Themba my daughter

Contents

Acknowledgements

I would like to express my sincere gratitude and appreciation to all the people who assisted in many ways to make this study a success.

- To my supervisor, Prof. DG Kourie. Thanks for the understanding, guidance, advice and patience you have give to me during this study.
- To my colleagues (UNISA), Mrs C Reyneke, Mr T van Dyk and Mr N Levine. Thank you for advice, support and your encouragement.
- To everyone who participated in the usability study.
- To my parents, Nomvula and Paulos. Thank you believing in me and your encouragement.
- To my wife Nana. Thank you for your love and patience.
- To my daughter, Sn'Themba (Muntshu kaBaba). Thank you for understanding when you do not have to.

Abstract

A tandem repeat is a sequence of adjacent repetitions of a nucleotide pattern-signature, called its motif, in a DNA sequence. The repetitions may either be exact or approximate copies of the motif. A minisatellite is a tandem repeat whose motif is of moderate length.

One approach to searching for minisatellites assumes prior knowledge about the motif. This approach limits the search for minisatellites to specified motifs. An alternative approach tries to identify signatures autonomously from within a DNA sequence. Several different algorithms that use this approach have been developed. Since they do not use pre-specified motifs, and since a degree of approximation is tolerated, there may be ambiguity about where minisatellites start and end in a given DNA sequence.

Various experiments were conducted on four well-known software packages to investigate this conjecture. The software packages were executed on the same data and their respective output was compared. The study found that the selected computer algorithms did not report the same outputs. The lack of precise definitions of properties of such patterns may explain these differences. The difference in definitions relate to the nature and extent of approximation to be tolerated in the patterns during the search. This problem could potentially be overcome by agreeing on how to specify acceptable approximations when searching for minisatellites.

Some of these packages are implemented as Academic/Research Software (ARS). Noting that ARS has a reputation of being difficult to use, this study also investigated the usability of these ARS implementations. It relied on literature that offers usability evaluation methods. Potential problems that are likely to affect the general usability of the systems were identified. These problems relate inter alia, to visibility, consistency and efficiency of use. Furthermore, usability guidelines in the literature were followed to modify the user interface of one of the implementations. A sample of users evaluated

the before- and after versions of this user interface. Their feedback suggests that the usability guidelines were indeed effective in enhancing the user interface.

Chapter 1

Introduction

1.1 Background information

Advances in the use of technology today allow for decoding the DeoxyriboNucleic Acid (DNA) structure of hundreds of organisms in a relatively short period of time. The decoded data is typically stored in databases for various reasons, including data analysis . Data analysis aims at encoding certain proteins and determining certain regulatory sequences. The comparison of genes to determine their relationship is thus enabled, both within the same species and between different species. Because of the large amount of data generated by improved DNA decoding techniques, it is difficult to manually analyse the generated data. Computer software offers a reliable alternative for the data analysis of sequenced DNA in a readable format. The application of computer science and information technology to the field of biology and medicine is called Bioinformatics.

DNA constitutes a sequence made up of four nucleotides – Adenine (`A`), Cytosine(`C`), Thymine (`T`) and Guanine(`G`). To a computer scientist, DNA is viewed as a string of `A, C, G` and `T` characters. These nucleotides form long sequences and may contain specific kinds of patterns in certain areas. Consider, for example, the following DNA sequence: `ACGTCT ACGTCT ATGTT`. Note that the pattern `ACGTCT` is repeated twice in the sequence. Moreover, the repeats are adjacent to each other. Such a repeated pattern is referred to as a Tandem Repeat (TR). The pattern `ACGTCT` is called the motif. Depending on the number of nucleotides in the motif, TRs are traditionally classified into three categories: microsatellites with motif length of less than 6, minisatellites with motif length greater than 5, but less than or equal to 100, and satellites of which the motif is greater than 100. Section 1.2 gives a formal definition of these terms as they will be used in this dissertation.

TRs are used during DNA profiling—a technique employed by forensic scientists for identity verification of individuals by relying on their DNA profile. This DNA profiling may be used during paternity testing, and for forensic medicine and population genetics [27, 33, 3]. TRs are also associated with human diseases like cancer and epilepsy [85, 5]. Clearly, then, detecting these TRs is important as it has social and medical implications to human existence.

There are a number of algorithms that automate TR detection and some are still under development. A selection of these will be investigated later in this dissertation. stressed Two types of TRs are distinguished: perfect and approximate TRs.

A Perfect TR (abbreviated to PTR) of motif M in string T is a sub-string in T consisting of two or more adjacent *exact* copies of M. The TR example given earlier was of a PTR. The detection of PTRs is of limited biological interest, due to biological events that occur in genetic sequences. Events such as mutation, trans-location and transversal events, render the gene copies imperfect [87]. Consequently, PTR detection algorithms are of limited value.

An approximate TR (abbreviated to ATR) of motif M in string T is a sub-string of T consisting of two or more adjacent *approximate* copies of M in T. Normally, a number k is associated with an ATR, indicating how far off it is from being a PTR. Broadly speaking, this number is based on the total number of non-matching nucleotides in each approximate repetition of M in T. As an example, consider the string T shown in the following table.

T =	A	C	G	T	C	A	T	C	G	T	C	A	A	C	A	T		A
Pos.	1						7						13		15		17	
M	A	C	G	T	C	A	A	C	G	T	C	A	A	C	G	T	C	A

T as given in the top row is a TR with motif M = ACGTCA. Three adjacent repeats of the motif string M are shown in the bottom row, and their starting positions are shown in the middle row as 1, 7 and 13. It is apparent that not all elements of T match exactly with the nucleotides of the three repeated motifs. In positions 7 and 15, there are so-called mismatch errors—one nucleotide in the motif has been substituted by another in the sequence T. Another error, a so-called deletion, occurs in position 17—one nucleotide in M is missing in the sequence T. Insertion—one nucleotide is inserted in the sequence T—is the opposite of a deletion. Insertion and deletions are collectively referred to as *indels*.

Note that the question of determining precisely which nucleotides do not match is not as simple as it might first appear. In general, non-matches can be ascribed to one of three different types of *errors*: mismatches, insertions or deletions. In certain cases, a given non-matching scenario can be explained in more than one way, each way involving a different selection of these error-types. A full elaboration of when these ambiguities arise and how to deal with them is beyond the scope of this present discussion and this dissertation.

In general, algorithms searching for TRs need to detect both PTRs and ATRs. One approach requires *a priori* specification of the motifs. In such instances, a list of motifs M is given as an input. The drawback of this approach is that unanticipated TRs embedded in the data may go undetected.

An alternative approach is employed by various algorithms in the field. In these cases, *a priori* knowledge about the motifs to be sought is not required. Instead, the algorithms employ various techniques for TR detection.

These techniques may not deliver identical results and consequently the consistency within detected TRs using these techniques is not certain. This dissertation will be investigating these uncertainties.

This investigation is limited to minisatellite detectors which detect *unanticipated* TRs. Henceforth, all references to minisatellite detection software should be construed as referring to the detection of unanticipated TRs.

Three types of software that are of interest in this study are Proprietary Software (PS), Open Source Software (OSS) and Academic/Research Software (ARS). PS is software that is developed mainly for commercial reasons. As this software is developed to make profit gains, companies developing this software usually allocate significant resources for the development of this software. These resources include money, specialised developers, etc. It is therefore important that the product produced i.e software and the source code, be protect from misuse by customers and competitors. This is done so that the returns on investments can be realised. PS is further discussed in Section 3.4.1.

OSS is software that is distributed with its source code, allowing users to modify and redistribute it for free. This software is usually developed to satisfy the needs of the developer. Therefore, OSS is usually developed under limited resources. It is for that reason that some of the features of the software that may be important to other users may be overlooked.

ARS is software developed by academics/researchers mainly for research purpose rather than for profit, as is the case with OSS. The software may be developed to test an algorithm, for example. Because of lack of profit incentives, ARS too is developed with few resources. This may also results in other parts of the software like the usability of the user interface being overlooked. However, an advantage of these software is that they are usually freely available. In some cases their source code is also made available, giving other researchers the opportunity to scrutinise the source code. Some of the ARS products are issued to the public as OSS products. Thus, every ARS has a potential of being an OSS product at the end.

Software packages under this study are either ARS products and/or OSS products and they are referred to as ARS packages in this dissertation. The primary reason for restricting this study to ARS packages has been pragmatic: the ready availability of these packages. However, there are many other benefits (and indeed, disadvantages) of OSS that have been widely discussed and debated. Various authors [63, 50] believe that by its nature, OSS is more beneficial than Proprietary Software (PS). As an example, Pande and Gomes [63] believe that OSS is fully customisable to meet one's needs and can be seen as more accountable because of the availability of the source code with the software. Potential users can easily scrutinise the source code as opposed to taking the word of PS salesman about what the source code does. Mittal and Singh [50] on the other hand are of the opinion that the availability of source code imposes a security risk on OSS. They also observe that OSS is not easy to use and end-users are not provided enough support via software documentation. The challenge in managing software documentation in OSS is due to frequent changes made in the software.

One of the recognised advantages of OSS is its potential benefits to developing countries, South Africa (SA) included. This is due, amongst other things, to the free nature of OSS—potentially cutting costs for developing countries which may be under financial constraints.

There are lots of similarities between OSS and ARS such as their development goal. They both follow similar development processes with same constrains such as the availability of resources. These similarities give ground for asserting that ARS could be viewed as a subset of OSS. With that in mind, it is generally acceptable that ARS would experience similar issues (such as usability issues) as does OSS. The usability of OSS has been widely studied and is discussed in Section 3.4.2 compared to ARS. With all that being said, issues experienced by OSS (as discussed in Section 3.4.2) will be taken to be applicable to ARS.

It is interesting to note that the recently approved The Criminal Law (Forensic Procedures) Amendment Bill B2-2009 [78] (known as the DNA Bill) provides a legal framework for investigative DNA profiling in criminal and intelligence contexts. This development points in the direction of increased general demand for suitable DNA expertise within SA and, by extension, also for software tools supporting DNA analysis. In this sense, the present study into ARS-based minisatellites detecting software has local relevance, albeit within a rather limited local user community.

One of the general criticisms made about OSS, which would be similar to that of ARS, is that it tends to be user-unfriendly. This is considered to be one of the factors holding back the wider adoption of OSS amongst users who do not have specific computing skills [53, 92, 90, 55, 21] and [13]. Hence, this study will also investigate the usability of minisatellites detecting software packages.

1.2 Definition of terms and concepts

The usage of some of the terminology in this field is not consistent as pointed out by De Ridder et al [16]. The terminology used by one author may have a different meaning to the next author. The list below presents the terminology that will be used throughout this dissertation:

Tandem Repeat (TR) is a string of nucleotides where the motif is repeated more than once. The repeated versions of the motifs are adjacent to each other. A TR may consist of only exact copies of the given motif, or may include approximate copies too.

A Perfect Tandem Repeat(PTR) is a TR with the exact motif repeated.

An Approximate Tandem Repeat (ATR) is a TR in which the versions of the repeated motif are not necessarily the same as the motif itself.

A Microsatellite is a TR with motif length $|M|$ such that $2 \leq |M| \leq 5$.

A Minisatellite is a TR with motif length $|M|$ such that $6 \leq |M| \leq 100$.

A Satellite is a TR with motif length $|M|$ such that $100 < |M|$.

Proprietary Software (PS) is software that is available at a cost and is licensed to a single or group of users . The software license does not permit redistribution or modification, and the software does not come with source code. *Microsoft Office* is an example of such software.

Free Software is software freely obtainable and distributable but may not come with source code or allow modification thereof.

Academic/Research Software (ARS) is free software that may not necessarily come with source code, but can be requested from its authors and modified by permission. Software packages under this study are ARS.

Open Source Software (OSS) is software that is free, distributable, comes with source code and hence, is modifiable. The only condition with OSS is that it should remain open source even after modification. *Libre Office* is an example of such software.

Henceforth in this dissertation, reference to a TR should be interpreted to mean a general reference to a tandem repeat string without making any specific assumptions about whether it is a PTR or an ATR.

1.3 Research goals, objectives and questions

This study investigates two problems. The first has to do with the consistency of the output of various ARS implementations that detect minisatellites. The second has to do with the usability of these ARS products. These matters are important for the following reasons.

It has already been suggested that detecting novel minisatellites may have a biological significance pertaining to genetic diseases, the evolution of species and forensic medicine. Various algorithms implement different techniques with the intention of discovering those novel repeats. It is, however, not clear whether these techniques produce the same results. To put it differently, the consistency of minisatellites detected by various minisatellite detecting algorithms remains untested.

Secondly, SA as a developing country has shown support for use of OSS in all its departments. However, OSS is being criticised for poor usability support [92, 90, 55, 21] and [13].

The overall goal of this dissertation is to address two research questions associated with the two problems mentioned above. This study aims at:

- focusing the user community's attention on the fact that there is a variety of ARS minisatellite detection software;

- raising awareness of the fact that the software offerings generally differ in output as well as in usability; and
- making empirical information available with regard to these differences so that informed decisions can be made when selecting a product.

The remainder of this section provides these research questions as well as the dissertation's specific objectives in a formalised, systematic fashion. An outline of the scope and general limitations of the work is also provided.

1.3.1 Research questions

1. The first research question may be formulated as follows:

   > Do the selected ARS implementations behave consistently with respect to the occurrence and positioning of minisatellites, even though they do not rely on information provided *a priori* about the positioning of minisatellites?

   In order to investigate this question, the following hypotheses were derived:

   1.1. H_1: Equivalent minisatellites (PTRs) are detected by different algorithms, when no mismatches and/or *indels* are allowed.

   1.2. H_2: Equivalent minisatellites (ATRs) are detected by different algorithms, when mismatches and/or *indels* are allowed.

   1.3. H_3: Non-equivalent minisatellites (PTRs and ATRs) are detected by different algorithms when the meaning of "approximate" assumed by the algorithms varies from one to the next.

Two minisatellites reported by two different algorithms are equivalent if they possess the following properties:

- The location of the minisatellites in a genetic sequence is the same—that is, the same offset and last index position.
- The motif sizes are the same.
- The nucleotides in the reported sequences are identical.
- The motif repeat count—i.e. the number of times a unit motif is repeated—is the same.

- The number and index positions of nucleotides—including errors—are the same.

In this study, the overall data sets detected by different algorithms are considered consistent if the algorithms are run on the same genomic sequence and the results consist of:

- the same number of minisatellites, and
- the minisatellites from different algorithms are equivalent, in the sense described above.

2. The second question this dissertation addresses is:

   > How usable are the software packages previously investigated to address the first question?

   To address this question, the following sub-questions are identified:

   2.1. To what extent do the selected minisatellite detecting academic/research implementations follow usability guidelines suggested in the literature?

   2.2. To what extent would implementing the usability guidelines as proposed in academic papers improve the usability of one of the ARS detecting minisatellites?

To answer the first sub-question, a technique for evaluating a design is used to evaluate selected Graphical User Interfaces (GUIs). An improved GUI is implemented, based on the results of the evaluation and software engineering guidelines. This GUI is subjected to usability testing by potential users. This will in turn answer the second sub-question.

1.3.2 Research objectives

By investigating the aforementioned questions, this dissertation aims to achieve the following objectives:

- To learn whether and to what extent there are similarities and/or differences in output minisatellites detected by the algorithms implemented in the ARS software under study that rely on an *ab initio* approach.

- To assess the usability of academic/research minisatellite detection software and consider how it could be improved.

1.3.3 Delineation and limitations

This study focuses on the output of algorithms that detect TRs, specifically minisatellites. Only algorithms that detect minisatellites without relying on knowledge from databases or libraries are investigated. This study intends to investigate the implementations classified as ARS.

In this study, a few implementations of minisatellites detecting software will be investigated, namely: Mreps, Phobos, TRF and ATRHunter. These software packages are classified as ARS.

Phobos and TRF implementations are also available as standalone programs with GUIs, making them suitable for usability testing in the context of this study. Due to the lack of stand alone GUIs on the other packages, this study will use another package called FireμSat in its usability investigation. This is a microsatellite-detecting package, but it is being enhanced to detect minisatellites as well. In this study, the original GUI of FireμSat was investigated and subsequently improved.

1.4 Brief chapter overview

The rest of this dissertation is laid out as follows:

- Chapter 2: This chapter indicates relevant ways in which TR detecting algorithms may be classified. Later in the chapter an overview is presented of a selection of algorithms that search for minisatellites. The selection includes those algorithms which are subsequently evaluated in the dissertation.

- Chapter 3: This chapter relates OSS to usability. It identifies the different types of software users, and elaborates on the meaning of software usability. It differentiates between PS and OSS, points to concerns about the alleged lack of OSS software usability and discusses what is being done to address these concerns. It should be remembered that OSS problems, especially the usability issues are also applicable to ARS due to the nature of their development processes.

- Chapter 4: The methodology that will be used to address the research questions as previously outlined in this chapter is presented in Chapter 4.
- Chapter 5 and Chapter 6: These chapters investigate the respective research questions using the methodology as described in Chapter 4. Thus, Chapter 5 focuses on the output produced by the ARS implementations under study and Chapter 6 focuses on the usability of these implementations. Each chapter later reports on the findings derived from the gathered evidence.
- Chapter 7: This chapter gives an overview of the work done for the dissertation, reflects on the findings and limitations of the study and identifies future research questions.

1.5 Concluding remarks

The importance of TRs has been explained. TR-supported forensic evidence could assist in solving criminal cases that have remained unsolved because of lack of evidence. There are many other TR-supported genomic identification needs—for example in paternity testing. Early detection of genes causing cancer and other genetically related diseases could help prevent risks associated with late treatment. In all these contexts, the detection of these minisatellites could be of significant value.

Chapter 2

Background: Minisatellite detecting algorithms

2.1 Introduction

The rationale behind searching for Tandem Repeats (TRs) in general, and minisatellites in particular, has already been discussed in Chapter 1. It was noted that various algorithms have been proposed and some implemented as executable programs to search for these TRs.

This chapter gives a background on minisatellite detecting algorithms, which will eventually lead to the discussion of the algorithms selected for investigation. The outline of this chapter is provided below:

- Section 2.2 focuses on TR algorithm categories. This section looks at how various authors have categorised TR detecting algorithms. Hence, the categories will be revealed into which those algorithms fall that have been selected for further study.

- Section 2.3 gives detailed criteria used in selecting algorithms for investigation in this study. At this point the algorithms that will be investigated in this dissertation will be re-emphasised.

- Section 2.4 provides a brief discussion on the broad approach used to detect TRs employed by each of the selected algorithms.

- Section 2.5 introduces a few algorithms that might have been selected for further investigation, but were not for reasons that are mentioned.

2.2 Classification of TR searching algorithms

The problem of searching for TRs has existed for years. Algorithms that use different approaches to solve this problem have been proposed by various authors. Attempts to categorise these algorithms based on the types of TRs they are searching for, and the approach they followed to search for these TRs, have also been conducted. In this section, two categorisation approaches are briefly explained, namely the approach used by De Ridder et al [17] and the approach used by Saha et al [70].

In their study of microsatellites detecting algorithms, De Ridder et al [17] categorised TR searching algorithms into three types, according to the type of TR it detects:

- *Microsatellite detecting algorithms*: These algorithms only search for microsatellites. One example is FireμSat [17]—which will later in this dissertation be evaluated for usability.
- *More general TR-detecting algorithms*: These algorithms may detect other types of TRs (microsatellites, minisatellites, and satellites) in their search. Algorithms investigated in this study belong to this category.
- *Other algorithms*: Algorithms that do not aim specifically at detecting TRs, but that nevertheless detect TRs indirectly in some manner.

Unlike algorithms that search for microsatellites only, the existence of algorithms that search exclusively for minisatellites *per se* is not evident. Generally minisatellite detecting algorithms include microsatellites and/or satellites in their searches.

Boeva et al [10] distinguish between two classes of minisatellites: those originating from microsatellites, and long minisatellites, for which it is hard to separate the sub-motifs. Minisatellites originating from microsatellites have a motif that can be broken down to a simpler motif, with length l, such that $2 \leq l \leq 5$. As an example, the sequence `ACGTACGTACGTACGT` may be regarded as a TR with motif length of eight, namely `ACGTACGT`, repeating two times, or as a TR with motif length of four, namely `ACGT`, repeating four times. By way of contrast, in the string `ACGTCTACGTCTACGTCT`, a motif of length six `ACGTCT` can be identified. This motif cannot be broken down into a much simpler motif. Thus in searching for minisatellites, it may be important to start by searching for microsatellites, so that microsatellites do not get categorised as minisatellites. This could be the reason why algorithms searching for only minisatellites, seem not to exist. In the light of the above, the algorithms that will be investigated in this dissertation will belong to *more general TR-detecting algorithms* according to the classification of De Ridder et al [17].

There are a number of more general algorithms that do not search directly for TRs *per se*, but that end up generating results that contain TRs. These TRs could be microsatellites and/or minisatellites. BLAST [11] and RepeatMasker [76] are two algorithms that have been identified by De Ridder et al [17] to be in this category. These kinds of algorithms will not be discussed further as they do not form part of this dissertation.

A second categorisation was noted in a study by Saha et al [70] whereby distinctions were made between TR detecting algorithms. These distinctions are based on two approaches followed by algorithms in searching for TRs. These approaches are:

***Signature and library* based algorithms:** These algorithms identify TRs, by comparing the string provided against a reference set of repeat sequences, i.e. against a library. RepeatMasker can be considered to be an example of a *signature and library* based algorithm [70]. A drawback of using this approach is that since it is based on *a priori* knowledge about the motif, novel TRs may not be detected. Algorithms based on this approach are considered to be beyond the scope of this dissertation and will not be discussed any further.

***Ab initio* based algorithms:** These algorithms identify repeating sequences without using reference sequences or known motifs during the repeat identification process. By using this approach, novel TRs can be detected and hence, new motifs may be discovered and be forwarded to the database for use by *signature and library* based algorithms. Algorithms based on this approach are the focus of this dissertation.

According to Saha et al [70], the process of detecting repeating sequences employed by *ab initio* algorithms can be divided into two stages. The first stage identifies possible repeating motifs, whilst the second stage sets the boundaries and extraction of these repeating sequences. Five techniques that serve for initial identification of possible TRs are briefly mentioned below as reported by Saha et al [70]:

Techniques for identifying repeating strings

- `Self-comparison:` In this technique, a subset of a DNA sequence is compared with the rest of the unclassified DNA sequences. ATRHunter [87] uses this approach.

- `k-mer or ‘‘word counting’’:` This approach views a repeat (as opposed to a tandem repeat) in a sequence S of length n as a sub-string w of length k that occurs more than once. This approach involves explicit enumeration of all frequently occurring exact sub-strings (called *k-mers* or ‘*words*’) in a given sequence(s). If the repetitive sub-sequence w cannot be extended without introducing errors, then it is called a *maximal repeat*. Two sub-strings of length k match if their sequences are identical. Mreps [37] and REPuter [42] are examples of algorithms based on this approach.

- `Spaced seed:` Instead of searching for perfectly identical matches of length k, as with the case of a *k-mer*, this approach conducts searches using a *seed* containing a defined level of tolerance for a variation in sequence identity and/or length. Saha et al [70] identifies Pattern-Hunter [47] as belonging to this group.

- `Dot matrix:` In this approach, a sequence is plotted against itself. Similar *k-mer* elements located within a user-specified range are detected and recorded. An example as noted by Saha et al [70] is Dot-plot [77].

- `Periodicity:` Unlike the aforementioned techniques, this approach transforms sequence data from the sequence (time) domain into the frequency domain, and performs analysis on the frequency data. Spectral Repeat Finder [73] uses a power spectrum generated from a Fourier transform[*] to identify auto-correlations of a sequence with itself. The high intensity peaks in the power spectrum of the sequence represent possible repetitive regions or elements.

The techniques mentioned above are merely used for identification of candidate motifs. A candidate motif can then be used to check whether subsequent strings form a TR. Saha et al [70] mention three techniques that algorithms used to extend and combine motifs into TRs. These techniques are:

[*]A Fourier transform expresses a mathematical function of time as a function of frequency, known as its frequency spectrum.

- `Clustering`: Algorithms implementing this technique identify the TR by further grouping together similar motifs to derive the final TR definition. That is, exact repeats that are close together are merged to generate a set of *merged repeats*. This process is usually guided by biological heuristics.

- `Graph representation with heuristics:` This technique involves building a repeat graph in which the vertices represent the recitative sequence or motif. The weighted edges represent the relationship among similar elements.

- `String extension:` This technique is used by most algorithms that employ the so-called *k-mers* approach. Maximal repeats are extend in both directions with the number of mismatches as a threshold.

A detailed description of how these motifs can be extended is presented by Saha et al [70]. Figure 2.1 shows the approach used by algorithms under this investigation.

This section has introduced two TR detecting algorithm categories and mentioned the categories into which algorithms that will be investigated in this study belong. These algorithms are "*more general*" in their search in terms of the classification by De Ridder et al [17] and "*ab initio*" in terms of the classification by Saha et al [70].

2.3 Selection of algorithms for investigation

As has been mentioned, this study is limited to minisatellites detectors which follow the *ab initio* approach to detect unanticipated TRs. There are many such algorithms or at least, many claim to be within this scope. A list of a few examples can be found at [82]. However, it should be noted that this list was mainly intended for microsatellites detectors. Nevertheless, some algorithms also detect minisatellites, as is mentioned on the site.

The initial approach in identifying algorithms for investigation was to list as many algorithms that search for TRs as could be found. This list was populated with the following algorithms:

- Phobos [48],

- Mreps [37],

- REPuter [42],
- FORRepeats [44],
- INtegrated Variable numbER Tandem rEpeat findER (INVENTER) [89],
- Exact Tandem Repeat Analyzer (E-TRA) [35],
- Tandem Repeat Finder (TRF) [8],
- Approximate Tandem Repeat Hunter (ATRHunter) [87],
- Spectral Repeat Finder (SRF) [74],
- Tandem REpeat in sequences with a K-meanS (T-REKS) [34],
- Burrows-Wheeler tandem repeat software (BWtrs) [15], and
- JSTRING [24].

This list does not include all the algorithms available and it was realised that investigating the whole list would not be practical within the scope of a dissertation. Therefore, criteria were needed to qualify an algorithm for investigation. The following criteria were formulated:

- The algorithm should search for minisatellites. After all, this is the type of TR this study is focusing on.
- The algorithm should not rely on an existing database or library of motifs. In other words, the algorithm should utilise an *ab initio* approach.
- An implementation of the algorithm should be available to run as standalone computer software.
- This software implementation should be freely available and easily accessible from the web.

After all of that, it was realised that there were still too many algorithms that fit these criteria. After due consideration, it was decided that because the selected few are readily available and are prominent in the literature, they would therefore make a good representative sample.

Thus, minisatellite algorithms can be grouped into three in the context of this dissertation:

Group 1: available and compared – These are algorithms that have been selected for further investigation in the dissertation. These algorithms are:

- Phobos,
- Mreps,
- TRF and
- ATRHunter.

Group 2: available and not compared – These algorithms are available but will not be further investigated in this study. In some instances, the algorithms have been investigated elsewhere, and in other instances, the algorithms were not accessible during the investigation.

- JSTRING was investigated on the study by Masombuka et al [4].
- INVENTER, T-REKS, E-TRA and BWtrs were investigated and reported on by Mokwana and De Ridder [51].
- Although REPuter is said to be available , attempts to access it have not been successful. A brief description of this algorithm is therefore presented below, but it not further investigated in this dissertation.

Group 3: not available – The algorithms in this group are not available as software packages for comparison and were therefore not investigated any further. However, the FORRepeats algorithm is an exception in that a short description is given below. This description is presented to give evidence about the existence of such algorithms for the sake of interested readers.

The algorithms presented here showed some similarities and differences in their minisatellite detection approach. ATRHunter followed the self-comparison approach whilst the other follows the k-mer approach. Although most of these algorithms take the k-mer route, they may use other different means in identifying these minisatellites. As an example, TRF uses a probabilistic model whilst Mreps uses TR extension. It is for that reason one may ask if all these approaches will yield the same results and this dissertation aims to investigate that. Figure 2.1 shows the grouping of the selected algorithms based on the classifications discussed.

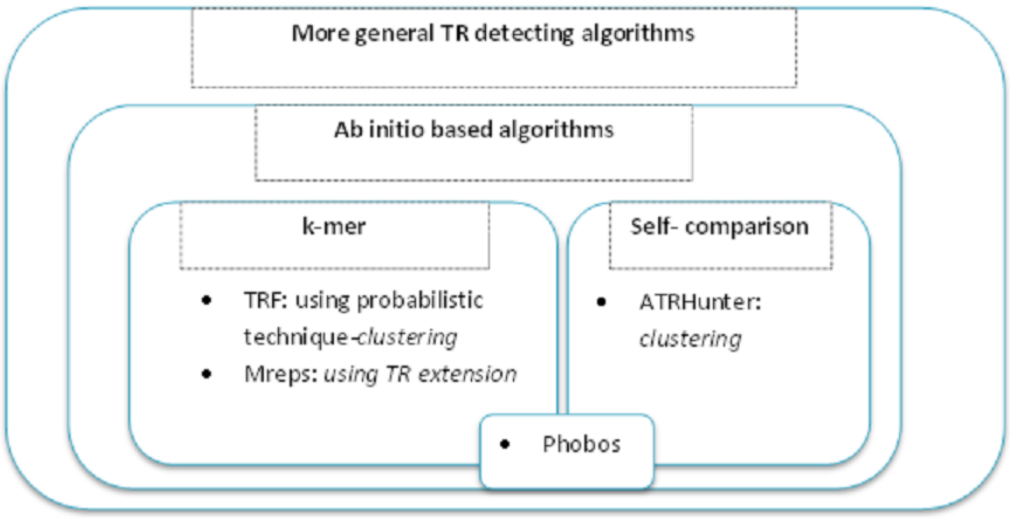

More general TR detecting algorithms
Ab initio based algorithms
k-mer
TRF: using probabilistic technique-clustering
Mreps: using TR extension
Self- comparison
ATRHunter: clustering
Phobos

Figure 2.1: Selected algorithms classified according to the approach and technique used by the algorithm. There is not enough information to the techniques used by Phobos, as is indicated in Section 2.4.1. This is the reason why parts of Phobos are placed in both k-mer, self-comparison and another part is outside them. This does not necessarily mean it belongs to all the techniques.

The next section gives a brief overview of Phobos, Mreps, TRF, ATRHunter, REPuter and FORRepeats algorithms. It should however be borne in mind that only the first four algorithms will be investigated further in this study.

2.4 Overview of algorithms selected for investigation

This section gives a brief overview of the selected algorithms for minisatellites detection. The criterion used to select the algorithms for investigation is as mentioned in Section 2.3.

It however should be noted that some details of the workings of TR detecting algorithms are not clear, as was noted by Leclercq et al [43], who focused on software packages searching for microsatellites. Therefore, the level of details presented here has been kept to the extent required in the context of this dissertation. Furthermore, the focus of the study is to cross-compare outcomes and usability of different implementations, rather than to explain exhaustively the internal details about each algorithm.

2.4.1 Phobos

Phobos uses what is called a *recursive alignment algorithm* [48] to detect TRs, including minisatellites. The developers of Phobos had not published a formal academic paper at the time of writing this dissertation, thus the deeper details of this recursive alignment algorithm behind Phobos and some of the details of their calculations are not well explained. The limited information about Phobos was obtained from the package's "*User manual*" [49] downloadable from the web together with the software. It is understood that this algorithm does not rely on a database or library of some sort in its detection, and hence, it falls under *ab initio* approach algorithms.

The Phobos algorithm checks each position in a DNA sequence and determines for each motif length, whether that position is a valid starting position for a match. After the valid starting position has been established, the detected repeat is extended in both directions as far as possible. Unfortunately, details of `how` Phobos determines this valid start position are not available.

A scoring scheme is used as an optimality criterion. Scoring is computed as follows: each match in the alignment gets a positive score of 1, (mismatch as well as insertion and deletion scores are restricted to negative numbers). A repeat is considered better than another repeat if it obtains a higher score. This optimality criterion is used to decide whether a repeat should be extended beyond a mismatch or an *indel* position or not [49]. Presently, the above is the only information that has been made available with regards to the optimality criterion.

Nevertheless, although the details on the algorithm employed by Phobos are not in the public domain, the implementation of this algorithm has been made available as a software package. A usability and comparative study of Phobos is therefore part of this dissertation, and it will be subjected to further investigation.

2.4.2 Mreps

Kolpakov et al [37] proposed the software package Mreps for TR detection. The algorithm underlying Mreps can be divided into two parts. The first part collects all repeating sequences using a combinatorial algorithm. The combinatorial algorithm considers that two adjacent sub-sequences or repeats, with the same motif length in a given sequence, are part of the same tandem repeat if they differ by no more than k mismatches. Sequences are compared and this process stops when the adjacent repeats differ by $k+1$ errors. More details regarding the combinatorial part of the algorithm follow below. The repeats collected in the first part are filtered by means of a heuristic treatment (a set of rules) in order to remove the repeats that are unwanted by the second part. The heuristic treatment is discussed later on.

The two parts of the algorithm are based on advanced string processing techniques. An overview of the two parts of the algorithm follows.

Combinatorial treatment

As mentioned above, the purpose of the combinatorial treatment is to identify all repeats occurring in a sequence. This is initially done by extending each repeat to the left and right without introducing errors. Such repeats are called *maximal*. A set of all maximal repeats in a sequence comprises all TRs occurring in this sequence.

To tolerate errors between repeated copies,the *maximal run of k-mismatch TRs* notation is used. Given an error threshold, k, a run of *k-mismatch* TRs of motif length m, is a string such that any sub-string of size $2m$ is a TR with at most k substitution errors [37].

For example,

`GCACAC ACACAC AG`

is a run of *1-mismatch* TRs of motif length 6. The corresponding TRs, each of motif length 6, are:

`GCACAC ACACAC`,

`CACACA CACACA` and

`ACACAC ACACAG`.

The computation of maximal runs of *k-mismatch* TRs of a given length has several limitations as pointed out by Kolpakov et al [37]:

- The parameter k, which is the number of mismatch errors allowed, should be specified beforehand. This implies that one has to have *a priori* knowledge of the motif one is looking for in order to be able to specify a proper number of allowed errors.

- There are certain artifacts of the definition of maximal runs of *k-mismatch* TRs, that produce unnatural so called '*side effects*' to the collected TRs. These side-effects are as a result of maximality run of *k-mismatch* TRs which implies that errors may be added to the end of a repeat in order to always reach k mismatches.

- The definition sometimes turns out to be too rigid: e.g. two long strings of a's separated by c should be viewed as two repeats, and not as the same repeat with a mismatch. In other words, a repeat can be seen as a repeat with multiple motifs in which one motif is better than the other(s). Indels not accounted for is another illustration of this rigidity.

- The reporting of insignificant repeats which may have no biological importance is also viewed as a limitation. These repeats may include repeats which have no obvious significance in the genome, like simply `AA`.

More details pertaining to the combinatorial algorithm can be found in [37]. The second part of the algorithm addresses these limitations and is discussed next.

Heuristic treatment

The repeats found in the first part of the algorithm are subjected to heuristic treatment to obtain the desired TRs. The heuristic treatment is completed in several steps that are explained below.

STEP 1: Trimming edges – This step removes errors that might have been added as a result of finding the maximal run of a *k-mismatch* TRs. For example, for $k = 1$ and with motif 2, `GACACAT` is *1-mismatch* TR repeating 2.5 times. However, the bases `G` and `T` in `GACACAT` could be deemed to redundant.

This step processes repeats in order to get rid of such artifacts. The '*cutting*' allows the removal of an edge element containing too many mismatch errors in it, and ensures that the remaining part of the repeat is still big enough to be meaningful. As an example, consider the repeat `GAAGGACAACGGACAGCGGACAATG` of motif size 7, found for $k = 2$. By cutting the three bases at each end, the repeat is reduced to `GGACAACGGACAGCGGACA`. Three bases are deleted (`GAA` at row 1 column 1, 2 & 3 and `ATG` at row 4 column 2, 3 & 4 below) from each end. The new TR is now repeating 2.7 times and has fewer mismatch errors. Row 4 shows a partial repeat whereby a repeat is formed by a fraction on a motif.

```
    1234567
1:  GAAGGAC
2:  AACGGAC
3:  AGCGGAC
4:  AATG
```

STEP 2: Computing the best period and merging – Another consequence of the definition of runs of *k-mismatch* TRs, is that the same region of the genetic sequence can be reported on for different motif lengths. E.g. for $k = 1$, the sequence `ATATATATATAAA` is computed as a repeat with
$motif = 2$ (`AT AT AT AT AT AA A`),
$motif = 4$ (`ATAT ATAT ATAA A`), and
$motif = 6$ (`ATATAT ATATAA A`).

Kolpakov et al [37] calculated what they called the '*best*' motif. This motif can be determined by computing the quality of a repeat as measured by the error-rate. This error-rate is given by:

$$error\_rate = \frac{error\_number}{length - m}$$

where *length* is the length of the repeat and *error_number* is the number of mismatches in the repeat, except that two mismatches formed by a nucleotide are counted for as one, if the mismatched nucleotides are the same. In other words, the following situation is accounted for one mismatch: $\underbrace{y \cdots x}_{p} \underbrace{\cdots y}_{p} \cdots$, where p is the motif.

The '*best*' motif, whose length is in the interval $[1...m]$, is the one with the smallest error rate.

The repeats in the above example will have error rates of:
$1/11 = 0.09$ for $m = 2$,
$1/9 = 0.11$ for $m = 4$, and
$1/7 = 0.14$ for $m = 6$.
Thus `AT` is regarded as the '*best motif*'. Hence it is referred to as the '`true motif`'.

After the *true motif* has been computed for each repeat, repeats that have the same motif and that are overlapping by at least two motif occurrences are merged into a single repeat. Overlapping repeats occur on top of one another. Consider the sequence `ACGTACACGTACACGTACTTTACTT`. The motif `ACGTAC` and `TACTT` overlap as shown below:

```
ACGTAC ACGTAC ACGTAC
                 |||
```

`TACTT TACTT`

When overlapping repeats are merged in this way, the parameter k no longer specifies the maximal number of mismatches between two adjacent copies of length m, but acquires a different meaning. As an example, consider the TR `ATATGG ATATAG ATAT` with motif size 6 detected for $k = 1$. Its true motif will be computed as 2 (`AT AT GG AT AT AG AT AT AT AT`[†]), even though there are two mismatches between adjacent `AT` and `GG`. Now instead, k measures the degree of fuzziness of computed repetitions and it is referred to as a *resolution parameter*.

STEP 3: Filtering out statistically expected repeats – This step filters out other repeats and retains those which are viewed as statistically significant. This is guided by a principle invoked in Bioinformatics, namely that only those repeats which have a small probability of being a random event can be biologically significant, as noted by Kolpakov et al [37]. A probabilistic model of a DNA sequence and a method for estimating the probability of observing a given repeat in a random sequence generated by the model could be used. For the purposes of Mreps, computer simulation and experimentally characterised repeats typically occurring in a '*random genomic sequence*' were used. This was because there is no corresponding theory, and elaborating on such a method is a non-trivial problem which is still largely open for research [37].

There are two reasons why a repeat can be statistically insignificant, according to Kolpakov et al [37]:

1. *The small length of the repeat.* A repeat that is perfect but too small. In other words it is very likely that such a repeat may appear in a random sequence.

2. *The high error rate of a repeat.* A repeat may be "*too noisy*", and therefore, indistinguishable from the "*background noise*". This means it will be difficult to see the difference between the repeated pattern and the raw sequence.

Two filters are used to eliminate the above statistically insignificant repeats. These filters are:

[†]Readers should note that the original source [37] of this example has an error in it and the error has been removed here.

- `length filter:` The length filter eliminates any repeat with a length smaller than $m+9$, where m is the motif length. Experiments showed that this filter eliminates short repeats occurring randomly very well.

- `quality filter:` The quality filter eliminates repeats by taking into account their error rate, their length and the resolution of the corresponding computed repeat.

STEP 4: Gathering the results – The final step constructs repeats found for different resolution values. The algorithm is iterated (STEP 1 to STEP 4) for all resolution values k, up to a certain value K that defines the final resolution level. Groups of collected repeats with the same motif are then processed again so that repeats overlapping with at least two motifs are merged into a single one.

2.4.3 Tandem Repeat Finder (TRF)

Tandem Repeat Finder (TRF) uses a probabilistic model to find TRs. The model is said to be probabilistic because it is based on percentage identicality. Two tandem copies of a pattern of length n are aligned by a sequence of Bernoulli trials[‡].

```
a g c t c a c t a g t a c a c a c a c t t
| | | | | | | |   | |     | | | | | | |
c g c t c a c t g g t - - a c a c a c t c
T H H H H H H H T H H T T H H H H H H H T
```

Figure 2.2: Two TR copies aligned, H is a match, and T is a mismatch or *indel*.

Figure 2.2 shows the idea behind the model. A match in the figure is shown as H for "*Heads*", and a mismatch, insertion or deletion is shown as T for "*Tails*". The *matching probability* is given by the average percent identicality between the copies. For example, the length of the top string in Figure 2.2 is 21, and the number of matches (the number of times H occurs) is 16. The probability of a match between the two example TR copies in the figure is therefore $\frac{16\times100}{21} = 76.2\%$. The second probability, the

[‡]A Bernoulli trial is an experiment whose outcome is random and can be either of two possible outcomes, "*success*" and "*failure*". The tossing of a coin can be identified with a Bernoulli trial, where, say,"*Heads*" is associated with success and "*Tails*" with failure.

indel probability, specifies the average percentage of insertion and deletion between copies.

The algorithm can be viewed as having two components. The *detection* component and the *analysis* component. The detection component identifies candidate TRs, whilst the analysis component attempts to produce an alignment for each candidate, and reports on TR statistics if the alignment is successful.

Detection component

The algorithm assumes that adjacent copies of any pattern will contain some matching nucleotides in corresponding positions. Thus, the algorithm looks for matching nucleotides separated by a common distance d. This distance is not initially specified, and to be efficient, the algorithm looks for a *run of k matches*, which is referred to as *k-tuple* matches [6].

This *k-tuple* is a window that contains k consecutive nucleotides. Thus, matching *k-tuples* are two windows of same length with identical contents. A Bernoulli model would produce a run of k '*heads*' if two *k-tuples* match, i.e. if they contain the same characters in the same order. As an example, if $k = 6$ a *run of k matches* will yield six heads—`HHHHHH`.

According to Benson [8], a list of all possible probes of *k-length* strings is kept. Within the DNA alphabet `A`, `C`, `G` and `T`, the number of possible probes of *k-length* strings is given by 4^k. Each probe p, is slid across the DNA sequence and a list, H_p, of occurrences of p in the sequence is maintained. Before every entry in position i in the list, the list is searched for earlier entries j of the same probe. Because i and j in the list are the indices of the matching *k-tuple*, the distance $d = i - j$ is a possible pattern (motif) size for a TR. Information about other *k-tuple* matches at the same distance is placed in a distance list D_d and this list is updated every time a match at distance d is detected. D_d can be viewed as a sliding window of length d which maintains the position of matches and their total.

Analysis component

As said, the contents of D_d are candidate TRs. The analysis component of TRF aims to produce an alignment for each candidate TR in D_d list. If the candidates in the distance list D_d passes the criteria tests—based on Bernoulli sequences and four distributions (refer to [7] for more information on tests criteria)—a candidate motif is selected from a nucleotide sequence and aligned with the surrounding sequence. If at least two copies of the motif are aligned with the sequence, the TR is reported. In addition to qualifying the candidate TRs, and producing the alignment which determines if the candidate TR will be reported, the analysis component also addresses the following issues:

- Multiple reporting of repeat at different pattern sizes: When several motif sizes are possible, TRF limits them to at most three motif sizes.
- Consensus pattern and period size: The candidate pattern from the D_d list is usually not the best pattern to align with the TR. In such cases, a consensus pattern is determined to improve the alignment.

2.4.4 Approximate Tandem Repeat Hunter (ATRHunter)

As with Mreps and TRF, the ATRHunter algorithm proceeds in two phases as well. These phases are the screening phase, which is performed first, followed by the verification phase. The screening phase identifies candidate ATRs of one of the types defined with respect to a scoring function φ [87]. The scoring function computes the similarity of two sequences—the higher the score, the more similar are the sequences. The following are types of ATRs as defined by Wexler et al [87] with respect to the scoring function φ:

A simple ATR is a concatenation of nucleotide sequences $T = T_1T_2 \cdots T_r$ for which there exists a sequence T_* such that $\varphi(T_i, T_*) \geq \eta$ for every $i = 1, \cdots, r$. Put differently, T consists of r (possibly) mutated copies of a *consensus motif* T_* with diversity limited by $\varphi(T_i, T_*) \geq \eta$. This definition may allow T_* to be different from each and every T_i. Alternatively, it may be required that T_* be equal to at least one T_i.

A neighbouring ATR is a concatenation of sequences $T = T_1T_2 \cdots T_r$ for which $\varphi(T_i, T_{i+1}) \geq \eta$ for every $i = 1, \cdots, r-1$. This definition allows the similarity between distant copies to be small.

A pairwise ATR which generalises the neighbouring ATR definition, is the concatenation of sequences $T = T_1T_2 \cdots T_r$ for which the similarity $\varphi(T_i, T_j)$ for every pair T_i and T_j is higher than the threshold η_{ij}. This η_{ij} limits the dispersal of distant sequences and is usually set to be a monotonically decreasing function of $|i - j|$.

The first definition is concerned with similarities between two sub-strings of sequences. The second definition focuses on two adjacent sub-strings, whilst the last definition is concerned with two remote sub-strings. Using the above model definitions, ATRHunter is capable of detecting the ATRs as per the given definitions. The processes involved in this detection are summarised below. A more elaborated discussion can be followed in Wexler et al [87].

Screening phase

The screening phase generates a list of candidate regions in the sequence that may contain ATRs. Thus, the screening phase identifies sub-strings which have an unusually high chance of being ATRs. Adjacent sub-string pairs in a repeat should be of a similar quality to an ATR. Therefore, this phase determines whether a sub-string of length t is similar to the adjacent sub-string of approximately the same length. This similarity between two adjacent sub-strings of length t is tested by comparing segments of length l of these two sub-strings. Every segment of length l (called *l-window*) in the first sub-string is compared with the corresponding *l-window* in the second sub-string. Details of how the parameter l is determined may be found in [87]. The result of this comparison is a vector entry, which indicates a match or a mismatch between corresponding pairs of nucleotides. This vector is called *q-quality* vector if the number of matches is at least $q \times l$, where $0 \leq q \leq 1$. The *score* $S_t(i)$ is the number of *q-quality* vectors produced by the comparisons of the *l-windows* in the sub-string of length t starting at position i. On the contrary, a *gap* $\Delta_t(i)$ is the number of vectors produced which are not *q-quality*.

The algorithm iteratively slides two *l-windows* towards the end of a sequence starting at position 1 and $t+1$. At each iteration, the two windows are slid towards the end of the DNA sequence whilst comparing their contents. The results of this comparison are candidate ATRs of length t. These candidate ATRs are kept in a vector of length l.

The candidate ATR identified should pass three *similarity criteria* in which a balanced criteria will limit the number of false positives and negatives. These criteria depend on motif length , the probability of match (P_M) between aligned nucleotides and the indel probability (P_I) between adjacent copies in an ATR. P_M is set to 0.25 if the distribution of the four nucleotide symbols (A, C, G, T) is uniform. Some form of biological model is used to determine P_I.

According to Wexler et al [87] a sub-string of length t starting at position i is a candidate ATR if and only if it satisfies these three criteria:

1. *Score criterion*: $S_t(i) \geq \sigma_t$, where σ_t is the maximum permitted score on motif length t.

2. *Continuity criterion*: $\Delta_t(i) \leq \delta_t$, where δ_t is the continuity threshold over the motif length t.

3. *Distance criterion*: Every *q-quality* vector in $S_t(i)$ is the result of a comparison between two *l-windows* whose *offset*—the distance of length t between the two *l-windows*—is at least 0 but not more than ${d^t}_{max}$. ${d^t}_{max}$ is the maximum distance between two *l-windows* of length t.

All candidate ATRs that came through the similarity criteria are still subjected for verification. The verification phase is discussed next.

Verification phase

During the verification phase, the list of candidate ATRs from the screening phase is validated to determine if the candidate ATRs are in fact plausible ATRs. Every type of ATR, as discussed earlier, is subjected to a different verification procedure. A two-phase procedure is performed for a pairwise ATR (Wexler et al [87] does not discuss the other verification processes):

- Firstly, the candidate ATRs and a sub-string are aligned to test if the alignment score passes a given threshold.

- Secondly, the ATR is extended by combining two-repeat ATRs into a single ATR containing more repeats. The motif length of the combined ATR is set to be the most common motif length among the original two repeat ATRs that generated it.

According to Wexler et al [87], the verification procedure does not allow overlapping ATRs with the same motif length. However, the algorithm may report ATRs with various motif lengths but starting at the same position. Nevertheless, if two ATRs have motif lengths for which one is a multiple of another, then the shorter motif is reported only if it scores significantly higher than the other ATR, or if it continues more to the non-overlapping (occurring outside one another) region . A certain statistical framework is used in determining the threshold, and further reading on it can be obtained in Wexler et al [87].

To summarise, ATRHunter has two phases. The screening phase identifies candidate ATRs. These candidate ATRs are not necessarily ATRs but are repeats which have a high probability of being an ATR. Some form of criterion that these ATRs should comply with is used to qualify a sub-string as a candidate ATR. The verification phase is then applied to verify the candidate ATRs. This phase accepts or rejects candidate ATRs identified in previous phase.

The algorithms that have been presented have some similarities in the way they search for TRs. These algorithms start by identifying initial exact TRs, which are later used as seeds (patterns) to find approximate TRs. Approximate TRS are identified by extending the seeds to the left and right whilst allowing errors within them. Although these algorithms possess similarities, the techniques they use differ.

The above algorithms meet the criteria that were presented in Section 2.3 for algorithms forming part of this investigation. In addition, these algorithms possess the Academic/Research Software (ARS) or Open Source Software (OSS) properties, making them suitable for investigation in this dissertation.

2.5 Overview of two unselected algorithms

As mentioned earlier, there are several algorithms that search for minisatellites, other than those discussed above. However, most of them do not meet the criteria that qualify them for further investigation in this dissertation. There are, however, two algorithms that almost qualified to be part of this dissertation, namely REPuter and FORRepeats. However, their standalone software programs were not available online for experimentation. This section very briefly surveys these packages for the sake of completeness. Additional information can be found in the cited material.

2.5.1 REPuter

REPuter reports various kinds of repeats. For the purposes of this dissertation only TRs will be considered. The discussion of the other types of repeats are beyond the scope of this report. However, the interested reader can consult [42].

Given a DNA sequence, REPuter computes all maximal repeats of length at least l, where $l > 0$. REPuter uses a variation of algorithms on exact and inexact string matching. REPuter are *inter alia* implemented by suffix trees.

A suffix tree is a data structure that represents the suffixes of a given string in a way that allows for a particularly fast implementation of many important string operations. The suffix tree for a string S is a tree whose edges are labelled with strings, such that each suffix of S corresponds to exactly one path from the tree's root to a leaf. Figure 2.3 depicts a suffix tree of string $S = aggc$. Gusfield [25] may be consulted for more information on suffix trees and their applications.

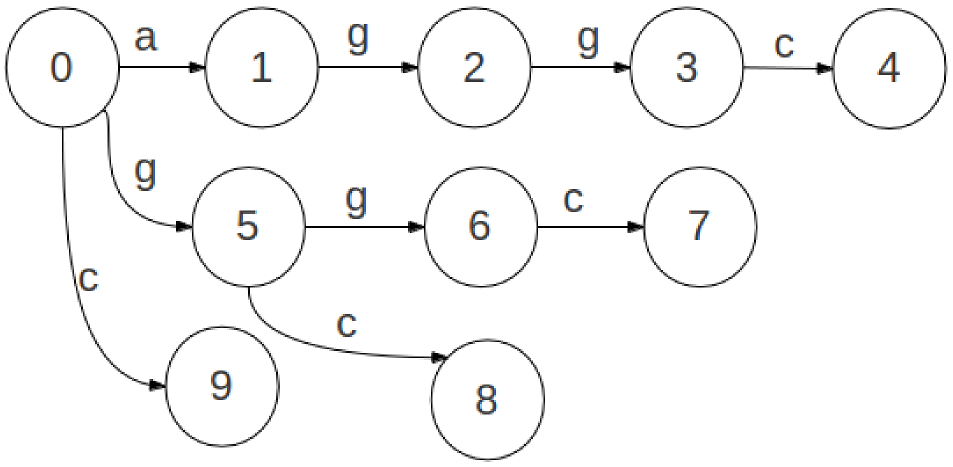


Figure 2.3: Suffix Tree of the string $S = aggc$ [30].

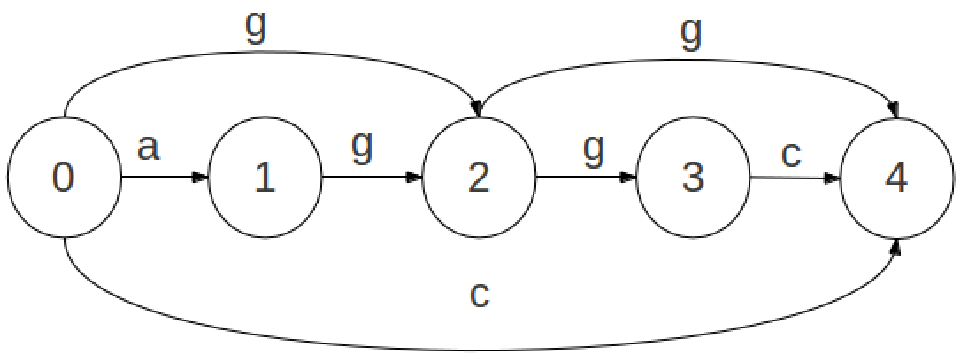


Figure 2.4: Factor Oracle of the string $S = aggc$ [30].

The REPuter algorithm finds TRs in two phases which are discussed next.

Phase 1: Finding exact repeats

Kurtz et al [40] use hashing as a method to compute maximal exact repeats. Hashing involves generating table entries for each DNA sequence w of length r and the positions $P(w)$ in string S, where w occurs. To find maximal repeats of length at least l, an exact repeat of length r is extended from left and to the right to check if it is embedded in a maximal exact repeat of the required length. Using the suffix tree for string S, Kurtz et al [40] directly computed the maximal exact repeats using the algorithm proposed by Gusfield [25].

In short, a table containing all the occurrences of a substring w is generated. From that table, w is extended from left and right until it can not be extended any further. Such sub-strings are maximal (they cannot be extended further) and are stored in a suffix tree data structure.

Phase 2: Finding approximate repeats

The mismatch repeat problem

A pair of sub-strings of equal length in a DNA sequence is a *k-mismatch*, if and only if the sub-strings do not occupy the same positions and their Hamming distance is at most k. The Hamming distance between two equal length strings is the number of positions at which they differ [42, 38, 39]. A *k-mismatch* repeat is *maximal covering* if it is not contained in any other *k-mismatch* repeat. The "*mismatch repeat problem*" is handled by listing all maximal covering *k-mismatch* repeats of length at least l that occur in a string S.

REPuter uses a Maximal Mismatch Repeat (MMR) algorithm to address the mismatch repeat problem. The MMR algorithm computes the seeds and tests for each seed whether it can be extended to a *k-mismatch* repeat. In other words seeds, are extended to the left and to the right allowing errors—mismatches in this case. A detailed discussion and the proof of the MMR algorithm can be found in Kurtz et al [41].

Indels: the difference repeat problem

The purpose of addressing the so called *difference repeat problem* is to extend the algorithm to allow for insertions and deletions. A pair of sub-strings is a *k-difference* repeat, if and only if the two sub-strings are not the same string and their edit distance is at most k. The edit distance between two strings of characters is taken to be the number of operations required to transform one of them into the other. To cater for *indel*s, another algorithm, Maximal Difference Repeats (MDR) is employed to compute seeds and try to extend them to *k-differences* repeats. The *k-differences* repeats allows for *indel*s. More details on MDR algorithm and its proof is presented in Kurtz et al [41].

The significance of repeats

Some of the detected repeats may not be considered to be biologically significant. The final step is to remove those repeats. For each repeat found by the algorithm, Kurtz et al [40] compute the value E as the number of repeats of the same length or longer, with the same number of errors or fewer than one would expect to find in a random DNA strings of the same length. This value E is used to assess the significance of a repeat found by the algorithm. A detailed explanation of how this $E - value$ is computed is presented in Kurtz et al [41].

2.5.2 FORRepeats

A drawback of using suffix trees as described above, is that they are not space economical. FORRepeats is a heuristic algorithm based on a data structure called a factor oracle, which is space and time economical [44]. Figure 2.3 and Figure 2.4 both show data structures implementation of string $S = aggc$. The suffix tree implementation is represented by Figure 2.3 whilst Figure 2.4 represents the factor oracle implementation. From the two figures, it is evident that the suffix tree representation has the most states, making it less efficient (in terms of space and time) than its counterpart.

Let p be a word of length m over the alphabet Σ. A word $w \in \Sigma^*$ is a $factor$ of p, if and only if, p can be written as $p = uwv$, where $u, v \in \Sigma^*$. The word u and v is a *suffix* and *prefix* of p respectively, if and only if, p can be written as $p = uv$ with $u, v \in \Sigma^*$ [1].

Factor oracle technology is an application of finite automata technology. The factor oracle of a word p of length m, 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:

1. It has exactly $m+1$ states. In a string $S = aggc$ with $m = 4$ the total number of states is $m + 1 = 5$ as illustrated in Figure 2.4.

2. It has between m and $2m - 1$ transitions. The total number of transitions in Figure 2.4 are far less than those of Figure 2.3.

3. It is homogeneous, meaning that for each state, all in-transitions are on the same symbol.

4. It recognises at least the factors of p and possibly more words.

5. It is acyclic automaton, which means that the states in the automaton are not connected in a closed chain.

The algorithm used by FORRepeats can be decomposed into two steps. The first step searches for exact repeats in a sequence using a factor oracle. The second step computes approximate repeats. This is done by extending each exact repeat to the left and to the right, allowing errors. This is realised until the similarity percentage between the two extended factors drops below a value fixed by the user. This similarity percentage can be computed using the Hamming distance. For the purposes of this dissertation, it is not necessary to discuss further details relating to FORRepeats. Readers who have more interest on the algorithm behind FORRepeats may consult [44].

The approaches employed by the various algorithms discussed have a number of similarities. For example, the approach taken by each can be divided into two parts. Firstly, possible seeds—in the form of PTRs—are computed. Secondly, these seeds are used to find ATRs. The differences lie in the techniques used to find and extend these seeds.

After selecting these algorithms, they were run on test data with the aim of comparing their results. This selection of algorithms for this empirical analysis was based on the availability and accessibility of the algorithm's implementation for experimentation. The results of these experiments will be addressed later in this dissertation.

2.6 Conclusion

This chapter introduced two approaches commonly used to search for TRs. The signature and library based approach requires *a priori* knowledge on regarding motifs pattern, whilst *ab initio* algorithms use techniques that require no knowledge regarding patterns to search for. Both approaches use algorithms to realise the goal of detecting novel TRs. However, using the *ab initio* approach allows for the detection TRs without any prior knowledge of their associated motifs.

Different algorithms which employed the *ab initio* approach for the detection of minisatellites were also briefly explained. The details of these algorithms have been discussed to the level deemed appropriate for the understanding of this dissertation.

Chapter 3

Background: Open Source Software and the relationship to Usability

3.1 Introduction

Apart from the fact that software should meet functional requirements, that is, it should do computationally what it is supposed to do, software should also meet certain non-functional requirements to be effectively used by any user. Chapter 2 reported on the functional aspects, that is *what* the software packages should accomplish—detect minisatellites.

Open Source Software (OSS) is often used as an alternative for commercial software—sometimes called Proprietary Software (PS). It is believed that OSS will be more beneficial to developing countries by reducing costs as compared to using PS.

It was mentioned in Section 1.1 that ARS share similar properties with OSS and some ARS products are also OSS products. It is in that regard that issues applicable to OSS in this dissertation should also be viewed as also applicable or related to ARS.

In this chapter the focus falls on the usability of ARS which detects minisatellites. The rest of the chapter will be presented as follows. Firstly, the usability challenges of OSS in general are investigated. The chapter starts by defining different kinds of users as they will be referred to in this dissertation, in Section 3.2. Section 3.3 explains what usability is focusing on and problems associated with it, with particular attention being paid to OSS. Section 3.4.1 gives a brief background on PS and its usability. This will be followed by a discussion in Section 3.4.2 of the OSS usability issues and suggested solutions to these problems by experts in the field.

The usability of software is related to its users. Therefore before discussing the concepts of usability in Section 3.3, a general overview of types of computer software users in the context of this study is provided in the next section, Section 3.2.

3.2 Computer software users

There are different types of software users within the computer community. These users can be classified according to their level of expertise in software development or use. For the purpose of this dissertation, developers and end-users are distinguished as follows:

- Developers/Programmers: These users are developers and/or implementers of algorithms. Sometimes they also use their self developed software. They are often referred to as *technical users*.

- End-users: These users are non-expert users of the software—i.e. they do not have high technical expertise. End-users may be everyday users of the software or new users who are unfamiliar with the software.

This dissertation is concerned with computational biologists—those who are experts in the field of computational biology, but who lack the technical skills of a programmer. Within the context of this dissertation, the term *users* will be used to refer to end-users including computational biologists.

3.3 Software usability

Non-functional requirements are features of a computer system software that are not directly related to the specific actions that a system should perform, but to the manner in which those actions are presented. In other words, non-functional requirements describe not *what* the software will do, but *how* the software will do it. FURPS is an acronym developed by Hewlett-Packard as noted by Sivess [75] to categorise non-functional requirements as:

- *Functionality*: which are requirements that are concerned with the features set, capabilities, generality and security of the system.
- *Usability*: addresses the human factors—user interaction, aesthetics—look and feel, consistency and documentation of the system.
- *Reliability*: involves the frequency or severity of failure, recoverability, predictability, accuracy and mean time between failure of the system.
- *Performance*: is concerned with speed, efficiency, resource management, throughput and response time of the system.
- *Supportability.* Involves the testability, extensibility, adaptability, maintainability, compatibility, etc, of the system.

The importance of non-functional requirements differs from one software system to another. For example, in some context the performance and reliability requirement of a software system may be more important than its usability. It is ideal to have a software system that complies with all specified non-functional requirements. This chapter only focuses on the usability requirement.

3.3.1 Alternative definitions of usability

Attempting to define usability, experts in the discipline have come up with various usability models. According to Hornbak [29] usability of software refers to the quality in use, or the capability to be used by humans easily and effectively. A number of definitions of usability have resulted from the different approaches which have arisen. These definitions enable researchers to evaluate the degree of usability of respective software packages. Proposers of these definitions include Shackel [72], Nielsen [56], Eason [20] and the International Organisation for Standardisation (ISO) [31, 32], an organisation aiming to set acceptable standards for various systems. In these definitions, various elements have been proposed as attributes that characterise usability and by which usability should be measured. In all the definitions, the attributes are not given any weighting to show their importance over other attributes.

The proposed usability models together with their attributes are discussed next.

Shackel's definition

In the model by Shackel, usability consists of four attributes, namely:

- *Effectiveness*: This attribute measures a user's performance in accomplishment of tasks.
- *Learnability*: Learnability of the software system measures the degree of learning required by the user who is not familiar with the software to accomplish tasks.
- *Flexibility*: This attribute allows adaptation to some percentage variation in tasks and/or environments beyond those first specified.
- *Attitude*: The attitude attribute measures user satisfaction with a system – within acceptable levels of human cost with regard to tiredness, discomfort, frustrations and personal effort.

Shackel's definition does not assign weights to these attributes because these weights may vary from project to project [45].

Nielsen's definition

According to Nielsen [56], software usability has five equally important attributes which should be equally incorporated during software user interface (UI) design. UI is the space where human and computer interact. These attributes are:

- *Learnability*: This component requires that the UI should be easy enough for new users to start using it, and get going.
- *Efficiency*: Once the UI has been learned, the user should find the UI easy to use to complete a similar task.
- *Memorability*: Once the UI has been learned, a not-so-frequent user of this UI should be able to remember how to use it without too much difficulty or relearning.
- *Errors*: The UI should prevent users from executing fatal errors, whilst users should be allowed to easily recover from non-fatal errors. Error rates should be kept minimal.
- *Satisfaction*: Users should enjoy using the interface of a computer system. They should not try to avoid using it.

The definitions of both Shackel and Nielsen stress that users should learn the new system, and after learning, they should be able to use the system to perform their tasks. Once the tasks are completed, users should be convinced that they have reach their goal. In addition to this, Nielsen stresses that non-frequent users should remember how to use the system after some time of not using it with relatively few errors.

Eason's definition

In Eason's definition of usability, the characteristics of the UI of the *system* under consideration, the characteristics of the *user* and the characteristics of the *task* will all determine the usability of the system [45]. These characteristics are usability attributes.

The UI should have the following characteristics.

System (User Interface) Characteristics :

- *Ease of use / Easiness*: Ease of use refers to the effort required to operate a system once it has been understood and mastered by the user.
- *Ease of learning*: This attribute measures the effort required to understand and operate an unfamiliar system.
- *Task match*: Herewith Eason refers to the extent to which the information and functions provided by a system match the needs of the user for a given task.

The characteristics of a task are as follows:

Task characteristics :

- *Frequency* is the number of times a task is performed by the user.
- *Openness* is the number of options that an interface offers for a task.

The user's characteristics are:

User characteristics :

- *Knowledge*: This attribute refers to the pre-knowledge the user has to apply to a task. It is easier to complete a task when a user has some knowledge about how to go about doing the task.

- *Discretion*: Discretion refers to the user's ability to choose to use (or not use) some part of the system at hand. As an example, someone may choose to learn to define one's own formatting styles in a word processor or choose to use tools in the package that give the same results.

- *Motivation*: The level of motivation a user has, in order to use a system to perform some tasks is measured by this attribute. Motivated users are more likely to start and complete a given task than less motivated users.

Eason's definition reflects the importance of considering the characteristics of the user as well as the characteristics of the respective tasks users perform. In contrast to that, the definitions of both Shackel and Nielsen do not give importance to the task at hand. According to Eason [20], usability should therefore not be measured solely by evaluating the UI.

International Organisation for Standardisation (ISO)'s definition

The international Organisation for Standardisation (ISO) is an international standard-setting body which is constituted by delegates from various national standards organisations. ISO provides two definitions of usability, namely ISO 9241-11 and ISO 9126. According to Raza [65], ISO 9241-11 was developed by Human Computer Interaction (HCI, *refer to Section 3.4.1*) specialists whilst the ISO 9126 was developed by software engineering experts. ISO 9241-11's attributes include the following:

- *Effectiveness* refers to the accuracy and completeness with which users achieve specified goals.

- *Efficiency* refers to resources extended in relation to the accuracy and completeness with which users achieve goals.

- *Satisfaction* refers to the comfort and acceptability of use of the system.

The ISO 9241-11 standard explains how usability can be specified or evaluated in terms of user performance view—*effectiveness* and *efficiency*, and perception of the system—*satisfaction*.

ISO 9126 defines the following attributes to measure usability:

- *Understandability* is the ability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use.
- *Learnability* is the ability of software product to enable the user to learn its application.
- *Operability* is the ability of the software product to enable the user to operate and control it.
- *Attractiveness* is the ability of the software product to be attractive to the user.

The two ISO definitions are complementary, in that they both define usability in a measurable design objective [23].

Table 3.1: Summary of usability definitions

	Shackel	**Nielsen**	**Eason**	**ISO 9241-11**	**ISO9126**
User performance (*Objective*)	Learnability -time to learn	Learnability	Easiness of learning		
	Learnability -retention	Memorability	Knowledge		
	Effectiveness - errors	Errors		Effectiveness	
	Effectiveness - time to complete a task	Efficiency		Efficiency	
			Ease of use		Operability
			Discretion	Task match	Understandability
	Flexibility		Openness		
			Frequency		
User view (*Subjective*)	Attitude	Satisfaction	Motivation	Satisfaction	Attractiveness

A summary of usability definitions is presented in Table 3.1 which is derived from Folmer and Bosch [23]. From the aforementioned definitions, it is possible to see that usability attributes can be divided into two facets:

- *Objective orientated*: These attributes measures user performance, and they include efficiency and learnability.
- *Subjective orientated*: These attributes are subjective in that they measure user perception, and include satisfaction, motivation and attractiveness of the user.

Measuring the subject orientated usability attributes is not easy because they vary from person to person. What may seem attractive or satisfying for one user, may not be applicable to others.

Furthermore it should be noted that there is a lot of ambiguity in the terms used for different attributes. As an example, in Shackel's terms, usability is influenced by the user's *attitude*, (which is *satisfaction* in Nielsen's terms). Nevertheless, these definitions largely overlap. These differences come from the authors' terminology preferences and their opinions on what they consider as meaningful usability attributes.

An understanding of attributes which contribute towards usability will not, of itself, result in a usable UI. It is important to know how these attributes may be incorporated into a UI for better usability. The next section looks at approaches that enable usability attributes to be built into a UI.

3.3.2 Design for usability

Folmer and Bosch [23] distinguish between two approaches that can be used when designing for usability—a *process oriented approach* and a *product oriented approach*.

Process oriented approach – user-centred design : User-centred design is a collection of techniques that specifically focus on providing and collecting facets of functionality that makes software usable. Usability is considered to be the design goal.

Product oriented approach – captured design knowledge : A product oriented approach considers usability to be an attribute formed by naming examples of product or system properties or qualities that influence usability. The use of this approach has resulted in a collection of design knowledge which consists of a collection of properties and qualities that have been proven to influence usability positively. A product oriented approach can be divided into three categories, namely:

- *Interface guidelines*: provide suggestions and recommendations for low level interface components (e.g. icons, windows, buttons, etc.).
- *Design heuristics and principles*: suggest properties and principles that have positive effects on usability.
- *Usability patterns*: describe best practises, good design and capture experience in such a way that it is possible for others to reuse this experience.

This section discussed the usability models as proposed by different experts in the field. The differences in these models to a great extent have to do with the differences in jargon that various authors use and prefer. Two design approaches on designing for usability were also briefly mentioned.

3.4 Usability in open source and proprietary software

The next section, Section 3.4.1 gives a brief introduction to PS and its usability. It is followed by a discussion of some of the issues associated with the usability of OSS, which are related to those of ARS, in Section 3.4.2.

3.4.1 Proprietary software and usability

Before the availability of desktop computers, computers were only used by highly trained programmers. The price of these room-sized machines could only be afforded by big organisations who could hire expert operators. Problems with the UIs were contended by technical users. As technology advanced computers got smaller and cheaper and more people were able to afford these machines in terms of space and money. At that stage the usability problems became more of an issue—unskilled users started to use computers. Software developers needed to develop software that can be used by non-highly trained users. Consequently, a new sub-discipline of Software Engineering emerged, namely Human Computer Interaction (HCI).

HCI is a discipline concerned with the design, evaluation and implementation of interactive computer systems for human use [45]. The aim of HCI is to enable developers/programmers to produce UIs that are usable by ordinary people. HCI is concerned with:

- Methodologies and processes for designing interfaces.
- Methods for implementing interfaces.
- Techniques for evaluating and comparing interfaces.
- Developing new interfaces and interaction techniques.
- Developing models and theories of interactions.

Involving HCI experts in PS development resulted in more usable software that is relatively easy to use by non-expert end-users. This can be seen in the high level of PS usage by corporations, small businesses and private users. Viorres et al [84] point out that the popularity of PS even extends to disabled users, the majority of whom prefer PS because of its greater accessibility and assistive technologies.

A detailed investigation of usability of PS is beyond the scope of this study and will not be discussed any further. In the next section, Section 3.4.2, OSS and its usability is discussed.

3.4.2 Open source software (OSS) and usability

Easy Internet access has led to the increased usage of free and open source software. Popular successful OSS products include Linux OS, Apache web server, Mozilla browser, GNU C compiler and MySQL DBMS [22]. It was originally believed that most users of OSS would be technically adept. The reason that led to this, is the development process followed by OSS programmers. However, non-technical users started to gain interest in OSS as well.

OSS and ARS developers usually develop the software for themselves. Schach [71] identified two informal phases that successful OSS (and/or ARS) goes through. Phase one entails that an individual developer builds the first version of the program and distributes it for free to anyone who might be interested. The development moves to the next phase, phase two, where the users become co-developers by reporting defects, whilst other users suggest ways to fix those errors or propose ideas of extending the program and implement those ideas. The unlimited expertise brought by these developers/users allegedly results in better quality code—Raymond [64] is of the opinion that "*given enough eyeballs, all bugs are shallow*". This means that it is easy to spot errors if more people rather than a few people are using and examining it. Thus, it is alleged that high quality code is in OSS products because defects are found and fixed more rapidly.

In their study on what motivates these developers, Hars and Ou [26] investigated the motivation of OSS developers and identified what they refer to as "*internal and external*" factors. The internal factors entail the person's hobbies and preferences. The external factors are rewards gained by a developer after he had associated himself with an OSS project. Self-marketing that promises future monetary rewards is an example of an external factor. Hobbyists and students are mostly internally motivated, whilst salaried and contract developers may seek to sell similar products or services and are externally motivated. According to Hars and Ou's study, external factors weigh more than the internal ones. However, Ye and Kishida [90] argue that the biggest driving force behind people who associate themselves with OSS communities is the process of learning.

The availability of minisatellite detecting ARS in this study for free for academic users, impliedissuadings that the authors of this software are probably both internally and externally motivated. Publishing academic research or the intent to do so, may be viewed as the external motivation for authors to be involved with ARS research projects. This is in turn a self-marketing strategy especially in cases where a non-free version of that software coexists.

Although the use of OSS is growing, the majority of computer end-users only interacts directly with PS. Various researchers have stated that usability issues are dissuading end-users from using OSS. Klencke's stance [36] is that usability issues with the UI of OSS, as well as the lack of usability engineering skills are major contributors to this behaviour. Nichols and Twidale [54] are also of the opinion that one of the reasons behind end-users' non-preference of OSS is linked to OSS usability.

Raza [66] agrees that usability is a complicated issue even in PS and that the issue is even worse when it comes to OSS. This could be due to the lack of HCI usability experts. The unusual OSS and ARS development strategy could also play a major role in contrast with PS development.

Viorres et al [84] noted that the "*bottom-up*" development approach followed by OSS development has an influence on the poor usability of OSS. The *bottom-up* approach gives lower priority to system modelling, user interfaces and other related issues, whilst focusing more on technical issues.

Nichols and Twidale [55] also provide a non-comprehensive list of features in OSS development that may contribute to the poor usability. They acknowledge that in many OSS products, developers are indeed users. This fact leads to products that have interfaces which are unusable by a group that is technically less skilled. They are also of the opinion that the non-involvement of usability experts is related to the incentives in OSS—the incentives for improved functionality are much higher than the incentives for usability. Feller and Fitzgerald [21] mention that traditionally there has been an overlap between OSS developers and users. The developers were the users of their own products and thus, more attention has been paid to the functionality rather than simplicity of the product. This tradition is slowly changing as more novice users are becoming interested in OSS because of the support that is being given by corporations like IBM.

It is clear that many authors agree that non-technical user involvement in OSS development has a strong influence on the weak relationship between OSS and usability. (See, for example, [92, 90, 55, 21] and [13].). Bodker et al [9] admit that the popularity of OSS products is threatened by its poor usability. Andreasen et al [2] conducted an empirical study and found that although OSS developers realise the importance of end-users, they still fail to prioritise usability issues. This is mainly because of limited understanding of usability, lack of resources like HCI experts or usability experts, evaluation methods that fit into the OSS paradigm, and the distinction between a developer and a user.

ARS users can relate to the above issues, like the unavailability of human experts who are will to work for no monetary gains. ARS is also developed with no end-user other than the developers themselves in mind. Section 3.4.3 looks at proposed solutions for OSS usability which would also benefit ARS usability.

3.4.3 Proposed OSS usability solutions

One potential solution suggested by Nichols and Twidale [55] involves using commercially developed user interfaces on OSS to improve its usability. While this solution can be seen as promising, Trudelle [83] believes that the two partner's interests may be in conflict.

Amongst the other potential approaches to improve OSS usability, Nichols and Twidale [55] also suggested a *Technical approach* which implies:

- the automation of the evaluation of the interfaces to compensate for the lack of HCI experts;
- greater academic and end-user involvement which encourages students involved with HCI to participate in OSS projects whilst also encouraging end user's to report usability flaws in a product; and
- more involvement of the HCI experts in educating OSS developers about the importance of usability.

In their empirical study, Raza et al [66] found that the user requirements, incremental design approach, usability testing and knowledge of user-centred design methods are key factors that strongly support OSS usability.

Terry et al [81] noted that some usability problems are common in OSS projects. The OSS and ARS community can improve its practises by being aware of what others are already doing to improve software usability. One way to accomplish this goal is to create a catalogue of the techniques that can help improve usability, along with instructions on how to get the most benefits from them.

Klencke [36] suggested standardisation, by adopting Graphical User Interface (GUI) standards, so as to reduce the inconsistencies that arise within OSS products. Klencke [36] also mentions using up-to-date documentation and tutorials to enhance end-user's learnability—ease of learning—of the system, encourage user feedback on usability on the software product and the use of a GUI instead of relying on text command interfaces. To avoid information overloading that may result when users report every usability problem with the software, Terry et al [81] proposed that different tools should be implemented for different user classes.

Similar to Nichols and Twidale [55], Klencke [36] acknowledges that it is important to involve usability experts. These experts may be involved by encouraging HCI students to be actively involved with OSS projects. Çetin et al [14] proposed that one way to get experts into OSS projects is to make developers more aware of basic usability principles. A model for learning usability inspection for non-expert evaluators was proposed by Zhao and Deek [91]. The objective of this is to transfer usability knowledge to non-expert evaluators, whilst actively performing inspection tasks.

OSS usability improvement efforts

There has been a gradual improvement in OSS usability, as many of the problems that are causing usability issues are now being addressed as follows:

- Several huge companies have started to support OSS usability by hiring HCI experts and providing other necessary resources that the OSS community does not have. These companies include IBM (Eclipse), Oracle (OpenOffice.org) and Novell (evolution, Mono, SuSE Linux) [36].

- The Human Interface Guidelines (HIGs) have been documented to give guidance regarding software application's look and behaviour in order to preserve consistency. GNOME[*] project's HIG is one example of many HIGs.

[*]GNOME is a desktop environment and graphical user interface that runs on top of a computer operating system.

- The OpenUsability project [61] was initiated to bring collaboration between developers and usability experts. By using the experiences of the members of the OpenUsability initiative, Reitmayr et al [68] have given suggestions on how usability should be integrated with OSS development.

- *Towards Open Source Software Adoption* (tOSSad) is a project funded by the European Union. The project was initiated to improve the outcomes of Free / Open Source Software (F/OSS) communities in Europe by sharing usability aspects and requirements among usability experts in order to help developers complete their OSS on time and under budget. This is done by producing the "*howtos*": tutorials, guidelines and how to carry out OSS usability tests.

- Research into and training about OSS has come into various university curricula and other training facilities. An MSc program in Turkey [62] and skills development programs in Syria and Middle East [80] all focusing on OSS, are two examples of such initiatives.

This dissertation entails an investigation into the usability of minisatellite detecting ARS, with the aim of addressing the usability concern of these software. Thus, addressing such concerns is likely to improve the adoption of such software.

3.5 Conclusion

The usability in all types of software has been shown to be important for software to gain maximum acceptance. Every developer's goal should be to develop a software products that will simplify the user's task and be the product of choice. Users should find it simple and enjoyable to complete their task, instead of avoiding it. OSS Authors in the field have tried to explain what is really meant by usability of software. Different qualities have been identified and measures proposed to evaluate usability. Usability is still regarded as something difficult to design by various researchers because of the subjective qualities it possesses.

Many researchers believe that OSS usability is poor, but should improve because it is in line with what happened to PS when desktop computers were introduced. OSS and indeed ARS has something to learn form PS when it comes to usability.

Having looked at the background information on minisatellites detecting algorithms and the usability issues faced by OSS and/or ARS, it is clear that the two issues need further investigation. Their importance to human life improvement, and the country's economy in terms of purchase costs, efficiency and licensing is apparent, and hence supports the significance of this study.

The next chapter details how the two aspects of this dissertation will be investigated.

Chapter 4

Methodology

4.1 Introduction

The primary goal of this study is to investigate two research questions as discussed in Section 1.3.1. These questions are:

1. Do the selected ARS implementations behave consistently with respect to the occurrence and positioning of minisatellites, even though they do not rely on information provided *a priori* about the positioning of minisatellites?

   The following hypotheses were proposed in order to answer the above question:

   a) H_1: Equivalent minisatellites (PTRs) are detected by different algorithms, when no mismatches and/or *indels* are allowed.

   b) H_2: Equivalent minisatellites (ATRs) are detected by different algorithms, when mismatches and/or *indels* are allowed.

   c) H_3: Non-equivalent minisatellites (PTRs and ATRs) are detected by different algorithms when the meaning of "approximate" assumed by the algorithms varies from one to the next.

2. How usable are the software packages previously investigated to address the first question?

   In order to answer this question, the following sub-questions were derived:

a) To what extent do the selected minisatellite detecting open source implementations follow usability guidelines suggested in the literature?

b) To what extent would implementing the usability guidelines as proposed in academic papers improve the usability of one of the ARS detecting minisatellites?

The first question will be investigated by setting up experiments, in which selected ARS packages (that is, Mreps, Phobos, TRF and ATRHunter) are executed on some sample genomic sequences. The outcomes of the experiment will be analysed to determine the similarities, and hence, the differences, between all the reported results.

To investigate the usability of the ARS implementations, Graphic User Interfaces (GUIs) are evaluated. The GUIs of FireμSat, Phobos and TRF will be considered. The evaluation will illuminate/answer the sub-question with respect to the ARS packages considered in this study, and within the constraints of the literature consulted.

After the evaluation has taken place, a new version of the FireμSat GUI—denoted as FireμSatPlus in this dissertation—will be designed. The suggestions on how OSS's usability could be improved, as discussed in Chapter 3, and the results of this evaluation will guide this design. Thereafter, usability tests will be conducted on the old and new versions by users who will perform identified tasks. Users' experience on the old and the new versions are measured in the form of a survey. The results of the survey will be used to answer the question in the positive or negative, depending on the outcomes of the findings.

The remainder of this chapter is set out as follows, starting by addressing the first question, Section 4.2 focuses on data detection. Section 4.2.1 introduces the research design to be followed and Section 4.2.2 presents the methodology used. The second question regarding software usability is addressed in Section 4.3 as follows: Section 4.3.1 briefly outlines the research design anticipated whilst Section 4.3.2 discusses the methodology within this context.

4.2 Minisatellites detection

The next subsections outline the research design and the methodology used to investigate the first question as articulated in Section 4.1. That is, this section focuses on the data detection question. Section 4.2.1 and 4.2.2 describe the research design followed and the accompanying methodology.

4.2.1 Research design

Mouton's [52] interpretation of a research design is that it is an outline of how one intended to carry out a study that is guided by certain research questions. The research design followed in this part of the investigation is a quantitative research. In order to explore the consistency of minisatellites detected by ARS implementation packages, this study will use an experiment as a research design. According to Hofstee [28] and Olivier [60], experiments are done to test a hypothesis or theory. This study has therefore identified three hypotheses to test in order to answer its research questions, as noted in Section 4.1. In the light of that, the experiments will be used to test these hypotheses. This technique was also used in a similar study conducted by Leclercq et al [43] focusing on microsatellites.

One of the challenges in implementing this research approach in the present context is to manage the large amount of data generated by the various algorithms. One way of meeting this challenge is to use a relatively small amount of input genomic data for sampling.

Figure 4.1 outlines the research design followed by this study. The next section focuses on the methodology that will be used within this research design.

4.2.2 Methodology

This section gives the instrument and how it will be used to investigate the first research question. An experiment will be used as an instrument to collect data.

Experiment

Four ARS implementations—Mreps, Phobos, TRF and ATRHunter—will be run on a desktop computer. Each software package will be given sample genomic sequences as input data. Appropriate settings for the various algorithms will be explored. On the one hand, settings for generating PTRs on each respective package will need to be determined. On the other, settings for generating ATRs on the respective packages will also need to be

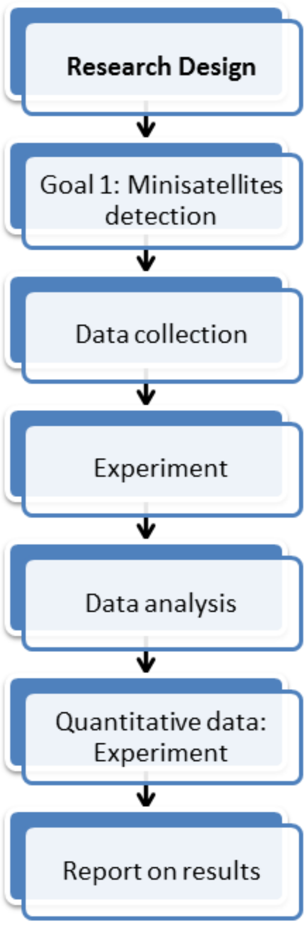


Figure 4.1: Research Design: Minisatellites detection

determined, but in such a way that the ATRs generated are as compatible as possible*.

Each algorithm will be allowed to be executed on this data sequence. The following algorithmic implementations will be investigated:

Mreps version 2.5.
Phobos version 3.3.10.
TRF version 4.00.
ATRHunter 1.00.

*It will be seen later why this is a non-trivial exercise and that it involved quite a lot of parameter-tuning.

All the above software packages are freely downloadable from their respective websites. Table 4.1 indicates the sample genomic data files that will be used to conduct the experiments and their respective sizes. The contents of each of these files is a string of characters representing DeoxyriboNucleic Acid (DNA) nucleotides. They look something like the following:

```
>sample test data
acgtgcacgtgcacgtgctgcacgtgcacgtgcacgaaaaaatttttttttttgtgtgtt
tttttgtgtgttttttttgtgttt
```

The first line—`>sample test data` in sample data identifies the sequence. This line is noted by the symbol `>` before the name followed by one or more lines of nucleotide sequences. This is referred to as a FASTA format[†]. The data files shown in Table 4.1 are all in FASTA format.

Each genomic file will be used to test one or more hypothesis. The process will be repeated for each and every software package under investigation. All the minisatellites of the respective packages reported will be recorded for later analysis.

Analysis

The results obtained from running experiments will be analysed manually by comparing minisatellites detected by the respective algorithms for each run of the experiment. Each comparison is based on the following minisatellite properties as outlined in Section 1.3.1:

- The index position: The occurrence of a minisatellite in a genetic sequence is at the same offset and last index position.
- The motif size: The motif size should be the same.

[†]A FASTA format is a text-based format for representing nucleotides sequences using single-character codes. The format begins with a single-line description, followed by (a)line(s) of sequence data

Table 4.1: Genomic data files.

Genomic Sequence	**Size**
Human X Chromosome	$\pm 50KB$
Jejuni genome	$\pm 1.60MB$
Swam genome	$\pm 8.30KB$
Sample test data	$\pm 1KB$

- The motif sequence: The nucleotides sequence should be in the same index position in the genetic sequence.
- The motif repeat count: The number of times a unit motif is being repeated, should be the same.
- The same TR: The length, the number and index positions of nucleotides—including errors—should be same.

Once the various algorithms have proposed their respective set of minisatellites, their outcomes will be analysed for consistency. This consistency will be based on the following facts:

- the same number of minisatellites, and
- the minisatellites from different algorithms are consistent.

The outcome of this comparison will then be reported and the final conclusions will be made based on the results of the experiments.

4.3 Software usability

Now that the process and procedures have been discussed that are to be followed when investigating minisatellite consistency once the minisatellites have been detected by the various packages, the focus of this section is shifted towards software usability. As has been mentioned, this study investigates two main questions; this section concentrates on the second question which relates to usability. Section 4.3.1 and 4.3.2 describe the research design and the underlying methodology, respectively.

4.3.1 Research design

To test the usability of the ARS implementations, a combination of evaluative and survey research methods will be followed. Hofstee [28] describes evaluative research as a technique that seeks to come to conclusions about an effect or success level of some happening or intervention. Olivier [60] refers to this as appreciative enquiry—a qualitative method that looks at the current situation and appreciates (understands) what already exists with a view to identifying possible areas for improvement.

A criterion is identified to evaluate the usability of the three GUIs mentioned earlier, i.e TRF, Phobos and FireμSat. The data obtained in this evaluation will be qualitative. Hofstee [28] warns about researcher bias when using an evaluative research method. A survey is adopted to detect any bias that may have been introduced. Obtained data from the survey will be quantitative. In summary, this dissertation will:

1. Evaluate the three GUIs, i.e. TRF, Phobos and FireμSat based on literature.
2. Build a GUI as an improvement of the FireμSat GUI.
3. Evaluate the new GUI and FireμSat GUI base on user experience.

In the context of this study, this intervention will be by constructing a proof of concept prototype. This will occur when the new version of FireμSat is constructed with suggested improvements. Olivier [60]'s stance is that prototypes are used by researchers as a proof of concept. In other words, researchers build prototypes to prove that a concept or an idea works. The results of the survey will determine whether using usability guidelines could improve open source minisatellite detecting software.

To summarise, this part of investigation uses the so called mixed-method which involves a combination of qualitative and quantitative methods. The qualitative method is used during appreciative enquiry, whilst the quantitative method will be used during survey. Figure 4.2 shows a simplified view of the research design followed in this part of the investigation. The methodology that will be used during this investigation is discussed in Section 4.3.2, which follows next.

4.3.2 Methodology

This section gives the instruments that are used to investigate the second research question. Two research instruments are used to collect the required data. These instruments are:

- Expert analysis[‡]: This instrument will be used during evaluation of the GUIs and it is explained below.
- Questionnaire: to collect data on user's perceptions of the GUIs.

Selected ARS GUIs are subjected to a usability evaluation. This is done by evaluating the GUIs through expert analysis. According to Dix et

[‡]Expert analysis is an evaluation of the system using a human expert or designer. One of the approaches to expert analysis is heuristic evaluation.

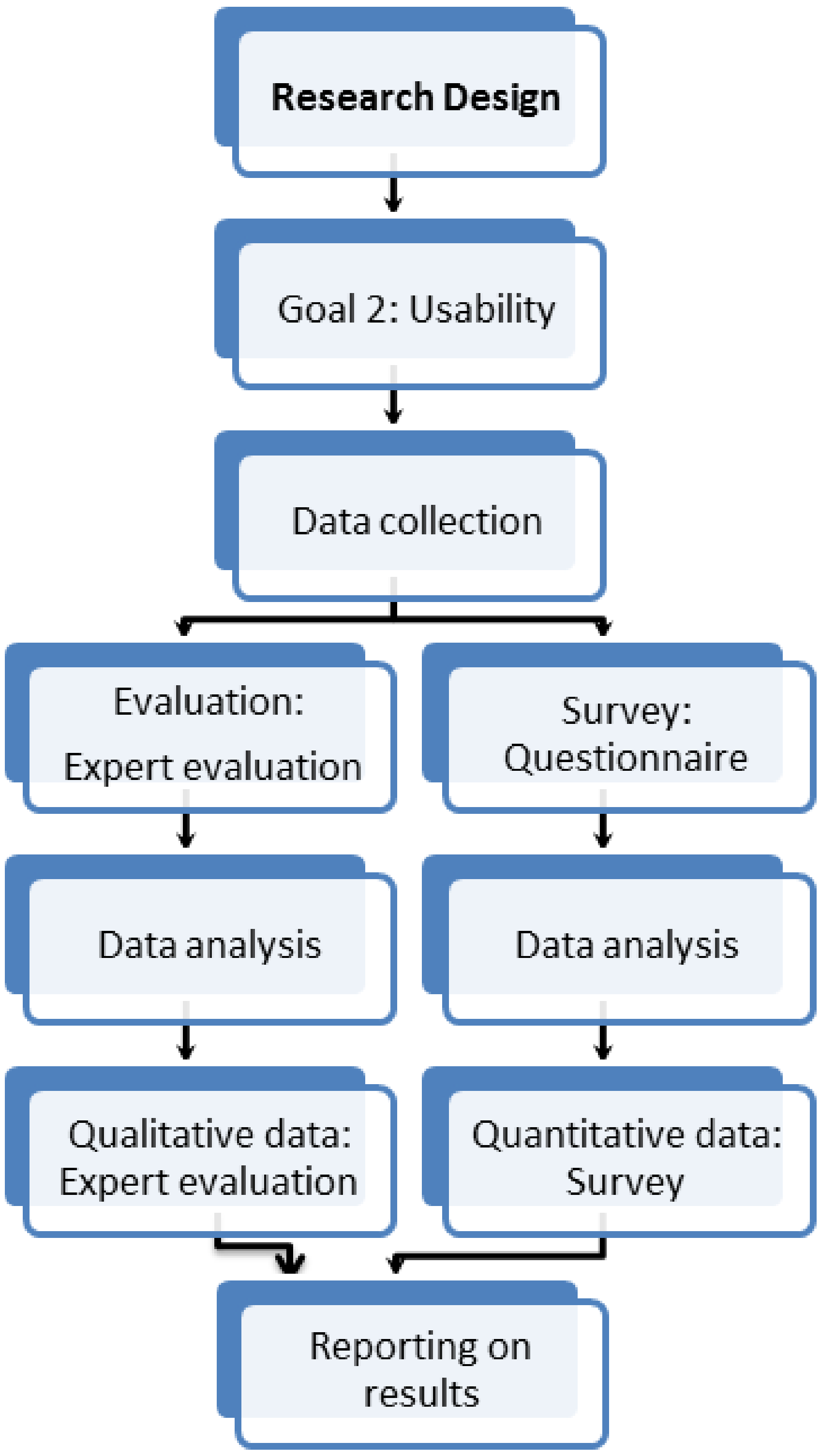


Figure 4.2: Research Design: Software usability

al [19], this is the first evaluation of the system that should be performed by designers to discover expensive mistakes that can arise when a design is left for later testing by users. Various evaluation methods which depend upon the designer, have been proposed. These methods take the design, and access the impact it will have upon a typical user. The basic intention is to identify areas that violate usability principles, and that are likely to cause usability problems. One approach to expert analysis is through heuristic evaluation.

Dix et al [19] refers to a heuristic as a guideline or general principle or rule of thumb that can guide a design, or be used to critique a decision that has already been made. Nielsen [56] proposed ten general heuristics which can be used to critique and hence guide a design decision. Nielsen's heuristics seem to integrate most of the existing heuristics. They are also widely used as trusted heuristics in the development of user interfaces. The author of this dissertation will therefore use Nielsen's heuristics to evaluate the selected GUIs. The data obtained as a result of this evaluation will be qualitative.

With possible areas that may experience usability problems identified, this study will intervene and attempt to address these problems by following possible solutions suggested by various OSS usability researchers (as discussed in Chapter 3) and design a new GUI. The new GUI will hopefully be an improvement of an existing package, FireμSat [17]—initially developed to detect microsatellites and now under enhancement to incorporate minisatellites [18] in its search. FireμSat has been developed like most ARS, with more attention being paid to the functionality of the software than the usability of the software. There is little work that has been done previously to improve the usability of FireμSat user interface. This dissertation will therefore improve the FireμSat GUI that was mainly built to aid the testing of the functionality of the software.

FireμSat and the improved GUI called FireμSatPlus within the context of this dissertation, are evaluated by conducting a usability test with users who mimic intended users. Nielsen [57] recommends using 3 to 5 participants in order to collect sufficient data. This study will therefore use two independent groups of 6 users. Appendix A gives a detailed explanation of how these users will be recruited and the procedure to be followed when conducting the usability test. User tasks will be identified and during usability testing, users will complete typical tasks. An existing usability questionnaire (Appendix D) is used to capture user's perceptions of the two GUIs. The questionnaire was used by Ssemugabi [79] during the study on usability evaluation methods. The advantage of using this questionnaire is two fold:

- its reliability has already been determined and
- it also measures usability based on Nielsen's heuristics.

Users are requested to complete identified tasks (Appendix C) on the GUIs and fill-out the questionnaire. Users' responses will be captured in the form on a five-point Likert rating scale[§] ranging from 'strongly agree' to 'strongly disagree'. Thereafter, the results are analysed and reported.

Analysis

Qualitative data obtainable from the first evaluation is in the form of a checklist in which the heuristic that the GUI adheres to is identified. Furthermore, the level of compliance and/or non compliance of the GUI to a specific heuristic should also be noted. From this data, it should be possible to conclude on the GUIs' usability or lack of it. The more a GUI complies with usability heuristics, the greater are its chances of being easily usable.

The qualitative data that results from the survey as Likert scale responses will be tabulated. Using descriptive statistics, the mean scores of the results will be calculated and be compared for the two groups. If the results show that there is a difference, a student's t-test will be used to measure the statistical significance of this difference. Student's t-test is a statistical technique used to measure if two groups' averages are statistically significant or occurred by chance. Olivier [60] gives a brief introduction on how this test is conducted and statistics books may be consulted for further details on this topic.   This analysis should draw conclusions about the extent to which traditional design for usability techniques in software engineering can be feasibly used for open source minisatellite detection software.

Limitations

A limitation regarding usability evaluation is the availability of minisatellite detection software package users. As the result, a few users who mimic real users are recruited to participate in the usability study. A sufficient number will be recruited in order to meet the minimum number of users required in usability testing as indicated by Nielsen [57].

[§]A Likert scale is a scale commonly involved in research surveys that employs questionnaires to scale response.

Ethical considerations

Various authors, including Hofstee [28] agree that all research should follow correct ethical procedures when conducting research. The participants were made aware that they are testing the software and that their responses will be treated in confidential. Users will be required to sign a consent form as included in Appendix B.

4.4 Conclusion

In this chapter, the analysis of the processes which will be used to investigate the consistency between minisatellites detected by various software packages was discussed. The guidelines on how the usability of these software packages will be evaluated were also considered. Using experiments and conducting usability testing on real users appears to be the best option in order to determine the usability of open source minisatellites detection software. In general, this study can be seen as mixed-method research.

The next chapter gives the results of the experiments conducted, while Chapter 6 gives the usability testing results.

Chapter 5

Minisatellite detecting algorithms: output comparison

5.1 Introduction

One approach used to discover novel minisatellites motifs is by using *ab initio* based algorithms. Without relying on some library or database of some sort, *ab initio* algorithms identify Tandem Repeats (TRs) from a given genomic sequence. Using techniques employed by *ab initio* algorithms would most likely increase the chances of discovering new minisatellite motifs and hence, new minisatellites.

The aim of this chapter is to investigate the consistency of the outputted minisatellites detected by different *ab initio* algorithms. This chapter entails a comparison of the data detected by a number of algorithms which detect minisatellites. The comparison is limited to selected algorithm implementations as indicated in Section 2.3. A summarised content of this comparison was presented in the Pattern Recognition Association of South Africa (PRASA) conference in 2010 [4].

All the algorithms are reported on in the literature (report on Phobos is limited) and are freely available from the Internet. The words software and algorithm in this dissertation may be used interchangeably to mean an executable implementation of an algorithm.

The overall goal of this chapter is to answer the question:

> Do the selected ARS implementations behave consistently with respect to the occurrence and positioning of minisatellites, even

though they do not rely on information provided *a priori* about the positioning of minisatellites?

This question is answered by investigating the three hypotheses proposed as follows:

- H_1: Equivalent minisatellites (PTRs) are detected by different algorithms, when no mismatches and/or *indels* are allowed.
- H_2: Equivalent minisatellites (ATRs) are detected by different algorithms, when mismatches and/or *indels* are allowed.
- H_3: Non-equivalent minisatellites (PTRs and ATRs) are detected by different algorithms when the meaning of “approximate” assumed by the algorithms varies from one to the next.

This chapter is laid out as follows: Section 5.2 discusses the research instruments, followed by an explanation of some of the parameters that are deemed useful for this investigation in Section 5.3. The investigation itself followed by discussion of results are presented in Section 5.5 and Section 5.6 respectively.

5.2 Research instruments

5.2.1 Test data

To investigate the question and the hypotheses, data files containing genomic sequences are used as shown in Table 5.1.

Table 5.1: Data files containing a representation of a genetic sequence.

Genomic Sequence	Size	Hypothesis
Human X Chromosome	$\pm 50KB$	H_1 and H_2
Jejuni genome	$\pm 1.60MB$	H_1
Sample test data (*as in Figure 5.1*)	$\pm 1KB$	H_2
swam genome	$\pm 8.30KB$	H_3

The data files are made available at `www.dna-algo.co.za` and are included on the compact disk (CD) which accompanies this dissertation. A simpler sequence was constructed to better analyse the behaviour of the algorithms. The contents of this sequence are as shown in Figure 5.1. The file is in FASTA format in which the top line is the name of the sequence preceded by symbol > and the line succeeding that represents the sequence itself.

```
>sample data
acgtgcacgtgcacgtgctgcacgtgcacgtgcacgaaaaaa
ttttttttttttgtgtgtttttttgtgtgtttttttgtgttt
```

Figure 5.1: Contents of sample test file.

5.2.2 Software

The four software packages under investigation are:

1. Mreps – `http://bioinfo.lifl.fr/mreps/`,
2. Phobos – `http://www.ruhr-uni-bochum.de/ecoevo/cm/cm_phobos.htm`,
3. TRF – `http://tandem.bu.edu/trf/trf.html` and
4. ATRHunter – `http://bioinfo.cs.technion.ac.il/atrhunter/`.

The following section briefly introduces the parameters for different software under investigation. The list of these parameters does not represent all available parameters for that software, but only those parameters that are necessary for successful completion of this investigation.

5.3 Software parameters

There are two types of parameters namely, the *searching* parameters and the *filtering* parameters. The searching parameters are necessary for the program to find all the TRs in a sequence, whilst the filtering parameters get rid of TRs which are not of particular interest to the user. In some cases, distinguishing between the two may require deeper knowledge behind an algorithm implementation, which is beyond the scope of this dissertation. As an example, to search for minisatellites with motif length of, say eight, algorithm A may start by searching for all possible minisatellites and then remove all minisatellites with motif length not equal to eight. Algorithm B on the other hand may immediately search for the required motif length with no need for filtering, which could be more efficient than algorithm A.

In the former case, the motif length has been used as filtering parameter, whilst it is used as the searching parameter in the latter case.

The next section discusses basic software package's respective parameters which are essential for the completion of this study. An exhaustive list of parameters may be found on the web pages of the respective software. In Section 5.3.1, the parameters of Mreps are discussed followed by parameters for Phobos in Section 5.3.2. Section 5.3.3 and Section 5.3.4 discusses parameters of TRF and ATRHunter respectively.

5.3.1 Mreps

Below are some of the parameters that a user may need to specify when searching for minisatellites when using Mreps. When these parameters are not specified, Mreps uses default values. Note that this list is not exhaustive, but shows only parameters that are important to successfully detect minisatellites as per the interest of this study.

`-from n` : This command processes the DNA nucleotide sequence starting from position n. This option is important when one has a very large sequence, but part of it should be ignored when searching for repeats, otherwise known as the flanking sequence.

`-to n` : This command indicates that the processing of the DNA nucleotide sequence should end at position n.

`-minp n` : The `-minp` is the parameter for setting the minimum motif length, referred to as "minimum period" by Kolpakov et al [37]. Only repeats with a motif length greater than or equal to n will be reported. Because Mreps can also search for microsatellites, this parameter will be important to filter microsatellites by setting n to 6. Only TRs with a minimum motif length of 6 will be reported. This parameter does not affect other repeats reported. That is, setting them minimum motif to 2 will yield the same minisatellites as when it is set to 6.

`-maxp n` : Indicates the maximum motif size of the repeat to be reported. Similar to `-minp` parameter, `-maxp` will disregard satellites when n is set to 100.

`-exp x` : The exponent indicates the number of repeated unit copies that should occur before the repeat is reported. It is given by the ratio of the whole repeat length to the motif length. Thus the parameter sets the algorithm to report only those repeats with an exponent greater than or equal to x.

-res n : The resolution parameter for setting the resolution of the program. It determines the amount of mismatches a TR is allowed to have. When set to 0, only perfect minisatellites are detected. The higher the value of n, the more errors (which may include *indels*) are reported.

All the parameters of Mreps relevant to the current discussion have been presented. In the next section, Section 5.3.2, the relevant parameters of Phobos are discussed.

5.3.2 Phobos

These parameters apply to Phobos software package:

-M exact : This is the option that should be selected to search for Perfect TRs (PTRs). Mayer [49] use the term *exact* repeat to refer to perfect tandem repeats (PTRs).

-M imperfect : Searches for Approximate TRs (ATRs) or imperfect repeats. This is the default search mode of Phobos.

-u <int> : This parameter sets the minimum motif length. The user should indicate the smallest motif length for which Phobos should search as an integer value. This parameter is equivalent to Mreps's `-minp` parameter. This parameter will allow the exclusions of microsatellites during a search if it is set to 6. Similarly to Mreps, this parameter does not affect other repeats reported. This means that, setting them minimum motif to 2 will yield the same minisatellites as when it is set to 6.

-U <int> : In contrast with parameter `-u <int>`, `-U <int>`, if represented with an upper case letter U, indicates the maximum motif length. (Mreps's equivalent of this parameter is `-maxp`, which would limit the search to minisatellites by setting its value to 100).

-indelScore : Penalty integer value given to *indels*, ranging from -4 to -6 with the latter being the default value.

-mismatchScore : Similar to `-indelScore` but, it is dedicated to mismatches.

-minScore : For a TR to be reported by Phobos, that TR should meet the minimum score which is set to 8 by default. As it was mentioned in Chapter 2, computation details of the score are not known.

It can be seen that some of Phobos parameters are similar to that of Mreps, whilst others are not the same—like the provision of scores.

5.3.3 Tandem Repeat Finder (TRF)

The format of the TRF's parameters is not similar to that of Mreps and Phobos in that these parameters are manipulated as a group than individually. These parameters are:

Alignment weights : Alignment weight for match, mismatch and *indels* is provided as a tuple of three integer parameters. Matches are given positive values whilst the other two (mismatches and *indels*) are treated as negative integers. TRF predefined the values of these parameters. In other words, the user selects from a given pre-selected values given as valid options. For example, the parameter values (`2,7,7`) indicates that:

- The match is given a positive value of 2. In all options, a match has a value of 2.
- The mismatch and *indels* are both treated as negative integers. The value of these integers can be either 3, 5, or 7.

The lower weights i.e. 3 and 5, allow alignments to accommodate more mismatches and *indels* in TR.

Maximum Motif Length : A maximum motif length of repeats to be detected. The program can find all repeats with a motif length between 1 and 2000. However, the output can be limited to some other range, say 100, in the case of minisatellites.

Alignment Score : A minimum alignment score to report repeat. When two or more TR sequences are aligned, the so-called alignment score is computed from based on the `alignment weights` selected. The details of how this computation is performed is beyond the scope of this dissertation. As with `alignment weights`, this *alignment score* is predefined and all TRs with or exceeding this predefined score are reported. The smallest score is 30 and the largest is 150. Higher scores are less permissive than lower scores, i.e. Most repeats will be reported on score 30.

The `Alignment weights` parameter is equivalent to the `-indelScore` and `-mismatchScore` parameters in Phobos, except that Phobos does not provide a parameter to set the score of a match. TRF could have done the same because the value of the match parameter is fixed, i.e. always a positive 2. Likewise, `Alignment Score` and Phobos's `-minScore` are also equivalent. In contrast, whilst Mreps and Phobos allows the minimum motif length to be manipulated, TRF does not have such a parameter. This implies that when searching for minisatellites, microsatellites will also be included in the report.

5.3.4 ATRHunter

ATRHunter follows a similar parameter format as the TRF. ATRHunter has the following parameters which can be set:

`Alignment weights` : This parameter is similar to that of TRF except that it has an additional parameter for terminal *indels*[*]. The *terminal indels* are treated as negative integers, which is the same as mismatches and *indels*. Similarly to TRF, a match score parameter is a positive integer value fixed to 2. Predefined values are also provided as options to choose from. These values varies depending on the definition used.

`Definition of an ATR` : ATRHunter supports different definitions of an ATR. The modularity of ATRHunter allows the user to decide which definition is more informative for one's research [86]. These definitions are based on the following kinds of ATRs:

- A simple TR,
- A neighbouring TR, and
- A pairwise TR.

Recall that the definitions of these ATRs were given in Section 2.4.4.

`Minimum Similarity level or Alignment Score` : The similarity level or alignment score computed for the ATR must meet or exceed this

[*]Terminal indels are insertions or deletions that occur at the beginning or end of a sequence.

specified value for the repeat to be reported. Depending on the definition on an ATR chosen,(Recall that in Section 2.4.4 definitions were given of three different kinds of ATRs: simple, neighbouring or pairwise.) ATRHunter uses either *minimum Similarity level* or `Alignment Score`. This score can be compared to the `alignment score` of TRF.

`Maximum Motif Length` : Only repeats with a motif length shorter than or equal to this value are reported.

Similarly to TRF, ATRHunter does not allow the minimum motif length to be specified. The manual exclusions of repeats with unacceptable motif length might be necessary at the end of the search.

Table 5.2: Summary of parameters

Parameter	**Mreps**	**Phobos**	**TRF**	**ATRHunter**
Motif size	Min. & Max.	Min. & Max.	Max	Max
TR Type	Resolution Par.	Perfect and Imperfect mode	—	—
Weights	—	—	Align. weights	Align. weights
Score	—	mismatch and indel score	Align. score	Align. score

Table 5.2 indicates a summarised set of parameter for the four software packages under investigation. As was noted earlier, this does not represent the exhaustive list of parameters available.

The only parameters set were those deemed necessary in the accomplishment of this study's objectives.

The table highlights the fact that Mreps and Phobos have the parameters to manipulate the minimum motif, whilst TRF and ATRHunter do not. Since this study focuses exclusively on *minisatellites*, it was considered relevant to establish whether the use of the minimum motif parameters in Mreps and Phobos would affect the nature or number of minisatellites detected by these packages. It was found that they did not—i.e. that the same minisatellites were detected by Mreps and Phobos for a given data set, whether or not the minimum motif parameter was set. Because of this consistent behaviour, the minimum motif for both Mreps and Phobos could be set to 6 to attain maximum efficiency without losing confidence in the results.

Another observation from Table 5.2 worth mentioning is the absence of the parameters directly changing the type of TR—perfect or approximate—in both TRF and ATRHunter. The TR type can be affected by adjusting the weights and scoring parameters. However, it is not possible to exclusively search for PTRs this way as these parameters only influence the chances or

PTR to be reported. Therefore, parameters in Table 5.2 were selected such that they increase PTRs chances in the reported results.

The above parameters will be adjusted depending on the investigation's goal on what kind of minisatellites (perfect or approximate) are of interest. These parameters will be set varying from low to high error tolerance. As a result fewer and shorter minisatellites with few errors should be detected, in contrast to long and highly imperfect minisatellites when parameters are set more loosely. The more errors allowed, the higher the number of detected minisatellites, can be expected.

5.4 Software Comparison

In order to investigate the aforementioned hypothesis presented in Section 5.1, three comparisons based on the results reported on by the different software packages were conducted. These comparisons were conducted by running a sample genomic sequence on the four algorithms and setting the algorithm's parameters accordingly. The observations of the reported data are then presented. The three comparisons are:

- Based on perfect detection of minisatellites – H_1. The algorithm's parameters are set not to allow mismatches and/or *indels*.
- Based on approximate detection of minisatellites – H_2. The parameters are set to allow as many mismatches and/or *indels* as possible.
- Perfect and approximate detection by TRF and ATRHunter using the same TR definition – H_3. Using TRF's definition of a TR, which is also implemented by ATRHunter, the parameters are set the same and the results observed.

The following observations were noted when conducting the comparisons:

- The number of repeats reported on by different packages.
- The similarity between repeats detected by different packages.
- Unique repeats reported on by different packages.
- Shortest and longest, motif length and TRs reported on by different packages.

The following section, Section 5.5, reports on the noted observations after the four algorithms were run on a genomic sequence and after the parameters were appropriately set.

A brief analysis of the minisatellites reported on by the respective algorithms follows, after the observations have been presented.

5.5 Observations

Table 5.3: Parameter settings for perfect tandem repeats (PTR) and approximate tandem repeats (ATR) for Mreps, Phobos, TRF and ATRHunter.

Search goal	**Algorithm**	**Parameter Value**
PTR	Mreps	$res = 0$
	Phobos	Search mode=$exact$
	TRF	weights(2, 7, 7), Alignment score (50)
	ATRHunter	weights(2, 7, 7, 7), Similarity level (0.5)
ATR	Mreps	$res = 6$
	Phobos	Search mode =$imperfect$, $indelScore, mismatchScore = -4$
	TRF	weights(2, 3, 5), Alignment score (30)
	ATRHunter	weights(2, 3, 5, 5), Similarity level (0.7)

After experimenting with various parameter settings and tuning them so that they produce competitive results, their setting in Table 5.3 were deemed approximate for this investigation. It is important that each parameter is tuned accordingly with respect to other parameters, especially on the software packages that do not have a parameter for ATRs and PTRs. In those cases, the weight parameter needs to balance with the alignment score parameters.

The parameter settings used for each comparison are shown in Table 5.3 for every respective algorithm. On the other hand ATR search parameters accommodate as many errors as the algorithm would allow. It should be noted that Table 5.3 indicates only those parameters required to produce PTRs and ATRs. Other parameters that do not influence the nature of TRs detected have been excluded from the table. Examples of such parameters include those affecting the minimum and maximum motif length. Phobos's scoring parameters were left to their default values where such parameter values are not indicated.

5.5.1 Comparison for perfect detection

> H_1: Equivalent minisatellites (PTRs) are detected by different algorithms, when no mismatches and/or *indels* are allowed.

Two genetic sequences were used as input data in the comparison of PTRs namely, *Jejuni* genome sequence and the *Human X Chromosome* as described in Table 5.1. The search parameters of the software packages were set in such a way that only PTRs should be detected. The relevant parameter setting are indicated on Table 5.3.

Table 5.4: The number of minisatellites reported for different motif lengths. The algorithms were first run on the *Jejuni* genome then on the *Human X Chromosome.*

Motif size	**Mreps**		**Phobos**		**TRF**		**ATRHunter**	
	Jejuni	**HX**[a]	**Jejuni**	**HX**[a]	**Jejuni**	**HX**[a]	**Jejuni**	**HX**[a]
6	1328	35	1328	35	1222	4	6	2
7	263	11	252	9	241	6	0	0
8	98	5	94	5	89	4	0	0
9	98	3	95	2	90	2	98	3
10	10	1	10	1	35	1	10	4
11	10	0	10	0	15	1	10	0
12	11	1	11	1	25	0	11	1
13	0	1	0	1	0	0	0	1
14	0	0	0	0	0	0	0	2
15	1	0	1	0	2	0	1	0
16	0	0	0	0	1	0	0	0
17	0	0	0	0	1	0	0	0
18-20	0	0	0	0	0	0	0	0
21	1	0	1	0	1	0	1	0
22-100	0	0	0	0	0	0	0	0
Total	**1820**	**57**	**1802**	**54**	**1722**	**18**	**137**	**13**

[a]Human X Chromosome.

The results obtained are summarised in Table 5.4 and the accompanying details of the reported results follows below.

Reported PTRs for the *Jejuni* Genome sequence

In this run, Mreps reported the highest number of repeats namely, 1820. Phobos reported 1802 repeats whereas TRF and ATRHunter reported 1722 and 137 repeats respectively. Furthermore, the following was noted:

- All the software packages reported on the longest TR in the data.
- All the software packages reported on the TR in the data that had the longest motif length, namely 21.

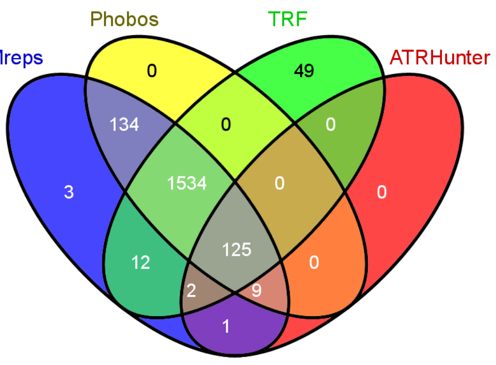


Figure 5.2: Venn diagram showing overlapping repeats from *Jejuni* sequence.

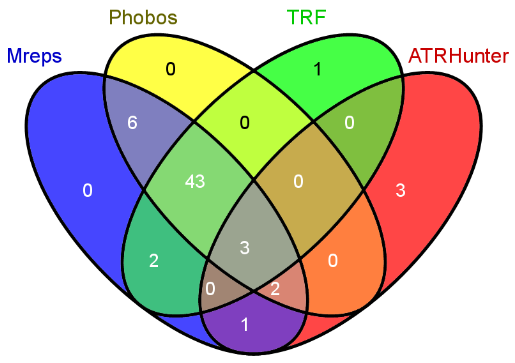


Figure 5.3: Venn diagram showing overlapping PTRs from *Human X Chromosome*.

In Figure 5.2 a graphical representation of detected repeats is provided as a Venn diagram[†]. The figure reflects the relationship of the number of repeats detected by one software package with one another. Thus, the total number of repeats that were detected by all the software packages is 125. From Figure 5.2, it is clear that Mreps detected 3 repeats that were not detected by any other software whilst TRF detected 49 repeats that were not detected by any other software.

Reported PTRs for the *Human X Chromosome* sequence

Figure 5.3 shows reported data similar to that of Figure 5.2 except that the numbers represent repeats detected within the *Human X Chromosome*. When closely observed, it appears that 3 repeats were detected by all the software packages. Although Mreps has detected the highest number of

[†]The Venn diagrams were generated using VENNY—a tool for comparing lists with Venn Diagrams [59].

repeats, it did not detect any repeats that were not detected by other software packages. TRF discovered 1 repeat that was not found by any other software package, and ATRHunter discovered 3 such repeats.

Mreps detected all the 54 PTRs found by Phobos, and found an additional 3. The situation is almost as good with respect to TRF and Mreps: Mreps found all except one of the 49 PTRs found by TRF, and also found an additional 9. The longest motif length reported on was 15, which was detected by both TRF and ATRHunter. From the reported data, it is clear that Mreps and Phobos detected more repeats when the motif length was small. TRF and ATRHunter detected more minisatellites than Mreps and Phobos when the motif length increases.

Analysis of algorithms for perfect detection

Clearly, there are some dissimilarities in the minisatellites report. To gain a better understanding on the cause(s) of this diversity, a short DNA sequence of length 84 (*see Figure 5.1*) was used as input data. Table 5.5 shows 5 different PTRs that were observed, and indicates the index position from where each PTR starts. From the table, the following can be noted about the software:

1. Only TRF did not report the repeat in PTR number 1. This could be because a longer repeat with perhaps better alignment i.e. better score, was reported in PTR number 2 instead of PTR number 1. A large portion of PTR number 1 lies inside PTR number 2.

2. Phobos did not report on PTR number 2 and 3. These repeats occur inside other repeats. It is possible that Phobos aims not to report on a repeat more than once. Thus Phobos avoids reporting on internal PTRs—PTRs occurring inside one another. That is, large parts of both PTRs number 2 and 3 occur within PTR number 1. In fact, according to the Phobos manual [49], in such cases only repeats with high scores are reported.

3. Only Mreps reported PTR number 3 as a potential repeat. As mentioned before, this repeat falls within another repeat—it is partly formed by the repeat in PTR number 2. All the other software packages use some scoring (*refer to Chapter 2 of this dissertation*) as a mechanism to qualify repeats to be reported. A repeat is first evaluated and if it scores above the stated alignment score, it is then

reported. In this case, this PTR appeared to have not met the score requirement.

4. Only TRF did not report PTR number 4. The reason behind that could be similar to that of item 1 of this list. Sequence 4 is embedded in PTR number 1 as reported by TRF (*See Table 5.6.*).

To summarise, Mreps reports all PTRs that are present, whist the other software packages filter out some. TRF and ATRHunter parameters allow for the possibility of longer repeats with errors. From these observations, it can be concluded that:

- Software parameters have an effect on the detection of minisatellites. If these parameters are internal to the algorithm, the user has no control over them.
- The nature of the repeat, e.g. a repeat occurring within another repeat, may differ depending on parameters used for that algorithm.

As results based on perfect detection have been presented, the output of the approximate detection are presented next.

5.5.2 Comparison for approximate detection

H_2: Equivalent minisatellites (ATRs) are detected by different algorithms, when mismatches and/or *indels* are allowed.

An extract from *Human X Chromosome* was used as the input data to determine how the data reported on by the various software packages differs in terms of the number of minisatellites reported. The parameters were set as loosely as possible, that is, the parameters for each software package were selected in such a way that the algorithm is given maximum latitude to decide that a given sequence of nucleotides should be construed as a minisatellite. Thus, each respective package was expected to report as many minisatellites as it was capable of identifying. Refer to Table 5.3 for the parameter settings applied for ATR search.

Figure 5.4 depicts the number of minisatellites detected by different software packages in relation to one another. From this figure, it is evident that TRF detected the largest number of minisatellites (186). Mreps detected 152 minisatellites. This is in contrast with the results obtained during the

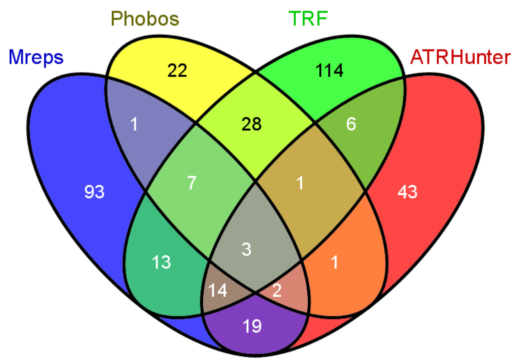


Figure 5.4: Venn diagram showing overlapping repeats TRs from *Human X Chromosome*. The repeats include both perfect and approximate repeats.

search for PTRs where Mreps detected the 53 PTRs (*49 for TRF, see Figure 5.3*).

Similarly, ATRHunter reported more TRs (89) than Phobos (65) during this search in which ATRs were allowed, whereas this situation was reversed in the previously described search for PTRs. During the search for PTRs, ATRHunter reported 9 PTRs and Phobos detected 54 PTRs (see Figure 5.3).

The software packages were capable of detecting only 3 common minisatellites. The number of *"unique"* repeats—minisatellites reported by one, and only one, of the software packages—increased drastically.

Analysis of algorithms for approximate detection

Using the same sample file in Figure 5.1 and running the software with parameters set to allow as many errors as they could, the outcome of this run is tabulated in Table 5.6.

Based on an analysis of Table 5.6, the following remarks can be made in the case of Mreps:

- In addition to PTRs detected on the previous run in Table 5.5, Mreps detected other new repeats when the resolution parameter was changed as reflected in Table 5.6. These repeats do not necessarily contain errors, for example, sequence number 2, 14, 17 and 19 are all perfect

Table 5.5: Perfect minisatellites detected by the four software packages.

Seq. #	**Start & end position**	**Motif length**	**Motif repeat rate**	**Motif**	**Sequence**	**software**
1	1 : 18	6	3	acgtgc	acgtgc acgtgc acgtgc	Mreps, Phobos & ATRHunter
2	4 : 33	15	2	tgcacgtgcacgtgc	tgcacgtgcacgtgc tg-cacgtgcacgtgc	Mreps, ATRHunter & TRF
3	10 : 27	9	2	tgcacgtgc	tgcacgtgc tgcacgtgc	Mreps
4	19 : 36	6	3	tgcacg	tgcacg tgcacg tgcacg	Mreps, Phobos & ATRHunter
5	48 : 82	12	2.92	tttttttgtgtg	tttttttgtgtg tttttttgt-gtg ttttttgtgt	all

Table 5.6: Approximate minisatellites detected by four software packages.

Seq. #	Start end Pos.	Motif length	Motif Rep. Rate	# errors	Motif	Sequence	Software
1	1 : 36	6	6.5	3	acgtgc	acgtgc acgtgc acgtgc acgtgc acgtgc acgtgc acg	Phobos & TRF
2	1 : 36	21	1.71	0	acgtgcacgtgcacgtgctgc	acgtgcacgtgcacgtgctgc acgtgcacgtgcacg	Mreps
3	1 : 42	21	2	6	acgtgcacgtgcacgtgctgc	acgtgcacgtgcacgtgctgc acgtgcacgtgcacgaaaaaa	ATRHunter
4	3 : 31	14	2	2	gtgcacgtgcacgt	g-tgcacgtgcacgt gtgcacgtgcacgt-	ATRHunter
5	4 : 33	15	2	0	tgcacgtgcacgtgc	tgcacgtgcacgtgc tgcacgtgcacgtgc	TRF
6	5 : 25	10	2	4	gcacgtgcac	gcacgtgcac- gtac-gtgcac	ATRHunter
7	10 : 27	9	2	0	tgcacgtcg	tgcacgtcg tgcacgtcg	Mreps & TRF
8	19 : 40	6	3.67	3	tgcacg	tgcacg tgcacg tgcacg aaaa	Mreps
9	36 : 82	23	2	10	gaaaaaattttttttttttgtgt	gaaaaaat-ttttttttttgtgt gaaaaaattttttttttttgtg-	ATRHunter
10	43 : 62	10	4	6	tttttttttt	ttgtgtgttt ttttgtgtgt ttttttgtgt	ATRHunter
11	43 : 84	10	4	5	ttttgtgttt	tttttttttt ttgtgtgtttt tttgtgtgttt ttttgtgttt	TRF
12	44 : 82	14	2.8	4	tttttttttgtgtg	tttttttttttgtg tgtttttttgtgtg tttttttgtgt	TRF
13	44 : 84	12	3.42	3	tgtgtttttttg	ttttttttttg tgtgtttttttg tgtgtttttttg tgttt	Mreps
14	47 : 66	13	1.54	0	ttttttttgtgtg	ttttttttgtgtg ttttttt	Mrpes
15	48 : 82	12	2.9	0	tttttttgtgtg	tttttttgtgtg tttttttgtgtg tttttttgtgt	Phobos & TRF
16	49 : 84	11	3.27	4	tttttgtgtgt	ttttttgtgtgt ttttttgtgtgt ttttttgtgttt	Phobos
17	60 : 78	13	1.46	0	tttttttgtgtgt	tttttttgtgtgt tttttt	Mreps
18	62 : 84	10	2.30	1	tttttgtgtg	tttttgtgtg tttttttgtg ttt	Mreps
19	63 : 75	9	1.44	0	ttttgtgtg	ttttgtgtg tttt	Mreps

repeats. However in this run, Mreps did not report all sequences which were previously reported i.e. sequence 5 of Table 5.5. Thus, when using Mreps different PTRs may be reported at different resolutions.

- The repeat in sequence number 5 of Table 5.5 was detected by Mreps when the resolution parameter was set to 0. When the resolution parameter was set to 6, Mreps did not picked-up this repeat as indicated in sequence number 15 of Table 5.6. Thus, different PTRs may be reported by Mreps when different resolution parameter settings are used.

- Some repeats were made longer by including a few errors in them. For example, repeat in sequence number 4 in Table 5.5 extended by allowing 3 errors, as shown in Table 5.6 sequence number 8.

Similarly, the following can be observed for Phobos:

- In some cases, Phobos combines adjacent repeats by allowing errors between them to form longer repeating sequences. For example, sequence number 1 in Table 5.5 and sequence number 1 in Table 5.6.

From the above observations, the following can be concluded about the algorithms which have been studied:

- The preferences of Mreps and Phobos are for short and perfect repeats. In contrast to that, TRF and ATRHunter are more prone to report longer and approximate repeats. This may be due to the parameters settings of TRF and ATRHunter which are more similar to each other as opposed to Mreps and Phobos.

- Although Mreps, ATRHunter and Phobos do report overlapping repeats, Phobos does not report an overlap if it is a multiple of another repeat [49]. TRF limits these repeats to at most three pattern sizes [6].

- Mreps reports partial repeats—repeats with a motif repeated less than two times. All the other software packages require that a motif be repeated at least twice before it can be reported.

It has been made clear that the algorithms in the study can detect different minisatellites on the same genomic sequences. These observations certainly contradict the aforementioned hypothesis—H_2. These differences seem to have been caused by different "*search logic*" used by these algorithms.

In order to ascertain if the parameter settings and/or the definitions of minisatellites are the only causes of the different minisatellites detected by the algorithms under consideration, a TRF definition of minisatellites will be used to test the last hypothesis – H_3. Section 5.5.3 explains the details of the test process.

5.5.3 Comparison between TRF and ATRHunter

The previous section showed clearly that different minisatellite searching algorithms under this study detect different minisatellites, even when presented with the same genomic sequence as input data. The possible cause for this behaviour is the different definitions of minisatellites employed by these algorithms, and hence, the parameters they use. What one algorithm regards as a minisatellite, may not necessarily be regarded as a possible minisatellite by another algorithm.

ATRHunter provides three different definitions of TR and three implementations which correspond to these definitions (Section 2.4.4). One of these definitions is claimed to correspond to TRF's definition [88]. To test H_3, an implementation of ATRHunter which corresponds to TRF's definition with the same parameter settings was used.

> H_3: Non-equivalent minisatellites (PTRs and ATRs) are detected by different algorithms when the meaning of "approximate" assumed by the algorithms varies H_3 from one to the next.

In order to do that, two comparisons on the same data were conducted. The first comparison used ATRHunter's default definition of a TR. The second comparison used the definition by TRF. The *swam* genomic sequence was used for this comparison as indicated in Table 5.1.

Using ATRHunter's default TR definition

Figure 5.5 indicates detected minisatellites with alignment weights set to allow as few errors as possible. It can be seen that each package reported the same number of minisatellites (10). However, jointly they detected 17 different minisatellites only 18% (3) of which were detected by both packages.

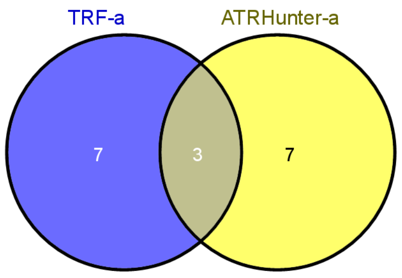


Figure 5.5: TRF weights(2, 7, 7) alignment score 30. ATRHunter (default definition) weights(2, 7, 7, 7) Similarity level 0.7.

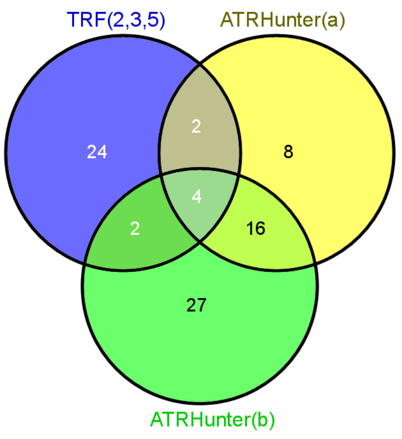


Figure 5.6: TRF weight(2, 3, 5) Align. Scr 30. ATRHunter-a(2, 3, 5, 5), ATRHunter-b(1, 0, 1, 0) and both have Similarity level. 0.7

To determine which role the parameters have in this observation, the parameters for the two algorithms were loosened to allow by all means every minisatellite a greater chance of being detected. Firstly, ATRHunter's alignment weights are set similar to that of TRF (*ATRHunter(a)*), then set to their maximum (*ATRHunter(b)*), as it appears in Figure 5.6.

TRF detected 32 minisatellites, whilst *ATRHunter(a)* and *ATRHunter(b)* reported 30 and 49 respectively. When parameters are loosened, more repeats were detected. Although the number of minisatellites common to (TRF and *ATRHunter(a)*) and (TRF and *ATRHunter(b)*) is 6 in both cases, 2 minisatellites are lost and 2 new are reported. This is probably due to increased motif sizes, as more errors are being tolerated.

Using the same (by TRF) definition

Figure 5.7 shows the relationship of TRF and ATRHunter using the parameters as highlighted in the figure. The following can be noted about using this common definition:

- The number of repeats detected by ATRHunter is 69, whereas TRF has detected 32.

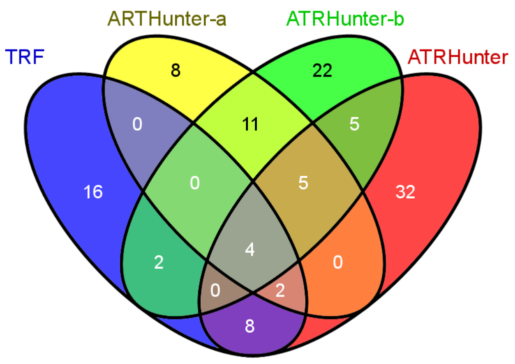


Figure 5.7: TRF weight (2, 3, 5) alignment score 30. ATRHunter-a and ATRHunter-b using the default definition weight (2, 3, 5, 5) and weight (1, 0, 1, 0) respectively with both Similarity Level of 0.7. ATRHunter uses the TRF's definition of TR with the weights set to (2, 3, 5, 5) and alig. Scr. of 30.

- ATRHunter has discovered 8 minisatellites in common with TRF that were not found by the variants ATRHunter-a and ATRHunter-b. In total, 14 minisatellites were detected by both TRF and ATRHunter.

- Two of TRF's minisatellites were detected by ATRHunter's default minisatellite definition (ATRHunter-b in Figure 5.7) that were *not* detected when using ATRHunter's TRF definition (ATRHunter in Figure 5.7).

- ATRHunter (TRF definition) and ATRHunter-b (default definition with very loose parameter settings) detected 14 minisatellites which are common to both.

From the above results, it is clear that TRF's definition enabled ATRHunter to detect more minisatellites than TRF and the two ATRHunter variants (ATRHunter-a and ATRHunter-b). Loosening the parameters increased the chances of detected minisatellites that were previously detected by TRF. This can be quantified. Of TRF's 32 TRs, ATRHunter-b detects 8 (i.e. 25%), and ATRHunter detects 14 (i.e. 44%). This is evidence that the definition on a TR does affect reported minisatellites.

Table 5.7: Similarities and difference between TRF and ATRHunter sequences.

Package	Start	Motif	Sequence
Both TRF & ATRHunter	75	tcact[a]attaaat	tcactattaaat tcacgattaaat
		tcacg[b]attaaat	tcactattaaat tcacgattaaat
	5679	aatgaaggg	aatgaaggg aatgaatgg[a]
			aatgaaggg aatgaatgg aatg[b]
	6497	ataatt	ataatt ataatt ata[a]
			ataatt ataatt ataa[b]
	6524	aaaagataaaaaaaaataag[b]	aaaagataaaaaaaaataag aaaa-[c]ataaaaaaaaataa
		aaaagataa[a]	aaaagataa aaaaaataa gaaaaataa aaaaaataa
	7233	ttttctc	ttttctc ttttctc t[a]
			ttttctc ttttctc tt[b]
	7922	ggtgct	ggtgct ggtgct ggt
TRF only	6526	aagataaaaaaaaat	aagataaaaaaaaat aaga-aaaataaaa
	6630	aaccttcgggag	aaccttcgggag aacc-tcggga
	6905	ttatttaa	ttatttaa ttatttaa t
	6974	tcttatatatat	tcttata-atat tcttatatatat
ATRHunter only	1095	gattcattt	gattcattt gattcattt
	3300	tgaaggaaccaagtt	tgaaggaaccaagtt tgaaggaaccaagtt
	5879	atactagtag	atactagtag atactaatag
	6740	acctacatt	acctacatt acctgcatt
	6857	ttaaaatac	ttaaaatac ttaaaagac
	6910	taattatttaa	taattatttaa taattatttta

[a]ATRHunter.
[b]TRF.
[c]*Indel*

Analysis of the results when the same definition is used

Table 5.7 shows the detected minisatellites. The upper part of the table shows those minisatellites detected by both algorithms, the middle part and the bottom part shows minisatellites detected by only TRF and ATRHunter respectively.

Starting with common minisatellites, those detected by both TRF and ATRHunter, the following was noticed:

- *Sequence 75*: Here, each package interprets the same sequence as a TR, but each assigns a different motif to the TR. Curiously, the motif used by TRF matches the last repeat rather than the first repeat (which is the conventional notion of a motif). Notice that the difference in motif choices does not affect the total number of matching errors. In each case there are only two repeats, and one contains a mismatch relative to the other, irrespective of which repeat is designated the motif. From this perspective, TRF could therefore have selected the first repeat as the motif, but did not do so for reasons that lie embedded at the coding logic.

- *Sequence 5679*: Both ATRHunter and TRF reported this TR as having the same motif. However, TRF appended *aatg* at the end of its TR sequence— a partial repeat sequence that does not fully cover the motif. This makes TRF's TR four nucleotides longer than that of ATRHunter.

- *Sequence 6497*: Once again, both ATRHunter and TRF reported this TR and used the same motif. In this case ATRHunter's last sequence group is *ata* whilst TRF's is *ataa*.

- *Sequence 6524*: In this case, the choice of the motif size is differs. ATRHunter's motif is 9 nucleotides, whilst TRF's motif is only 19. This leads to a sequence being regarded as a TR, but viewed as having different characteristics by the different packages.

- *Sequence 7233*: The TR identified by both ATRHunter and TRF at this offset has similar characteristics to that identified at 6497: in each case, the TR has the same motif. ATRHunter's last sequence group is simply *t* whilst TRF's is *tt*.

- *Sequence 7922*: At this offset, ATRHunter and TRF report exactly the same minisatellite.

The above minisatellites have similar starting positions, but they differ in one way or the other. These differences may result not only from the motif, but also on the repeating sequence itself. A trade-off between longer TRs (with more errors) and shorter TRs (with fewer errors) remains important. Terminal minisatellite nucleotides may in turn affect the offset position of the subsequent minisatellite, otherwise resulting in overlapping TRs.

The table also highlights minisatellites that were detected by TRF but not by ATRHunter (at offsets *6526, 6630, 6905* and *6974*) and vice-versa (at offsets *1095, 3300, 5879, 6740, 6857* and *6910*). Although some minisatellites were detected by certain software packages and totally missed by others, some minisatellites were embedded within one another. As an example, the minisatellite detected by TRF at starting position 6905 is part of the minisatellite reported by ATRHunter at position 6910.

Certainly, the above results belie the hypothesis proposed earlier. From the above observations, it is evident that ATRHunter detected different minisatellites, although ATRHunter claimed to use the same minisatellite definition as TRF. One possible explanation for this difference, may again point to the use of parameters. Although ATRHunter and TRF define the weight and alignment score similarly, ATRHunter's weight parameter is not exactly the same as that of TRF. TRF's weight parameter is a tuple of three whereas ATRHunter's weight parameter is a four-tuple. The fourth weight is for *terminal indels*—*indels* at end positions.

5.6 Discussion of results and conclusions

The results from the first comparison have shown hypothesis H_1 to be false. From these results, it is possible to conclude that Mreps and Phobos are better as ATR detectors, than TRF and ATRHunter—this is due to parameter settings that can favour longer TRs for the latter group. Therefore, Mreps and Phobos are arguably more suitable to be used as *seed* generators—that is, be used to find new motifs that can be used by other algorithms, including those that rely on minisatellite databases for their search.

Similarly, the second hypothesis H_2 was falsified by the comparison of approximate TR detection. The reported results revealed that no matter how loosely the parameters are set, differences between minisatellites reported still exist. In fact, the more errors that are allowed within a minisatellite, the more divergences between minisatellites identified by the different packages.

The possibility that this is as a result of the different minisatellite definitions used by these algorithms was investigated using hypothesis H_3. The outcome of this investigation did produce enough evidence to support hypothesis H_3. In this investigation, ATRHunter produced more repeats than its rivals, TRF and ATRHunter (using the default definition), and this is supported by experimental results conducted by the developers of ATRHunter [88]. Although ATRHunter algorithms claim to use the same definition of a TR as TRF, this definition was implemented differently from that of TRF i.e. the inclusion of *terminal indels* by ATRHunter. This somehow has resulted in these differences. Other implicit parameters that are internally employed by the algorithms in their search may also have something to do with this difference.

Although the high-level definition of minisatellite used by investigated software packages more or less correspond, there are differences in the actual minisatellites detected. These differences mainly result from the decisions about the extent to which a minisatellite is allowed to stretch (maximally, or in the case of Mreps, minimally) and still be regarded as a minisatellite. Depending on the algorithm, this maximal stretching may affect whether the subsequent minisatellite will be reported or not. Error tolerance of the algorithm is another cause of these differences. This can be seen in an algorithm's preference for PTRs versus ATRs. The sensitivity of parameters influencing this tolerance differ from one algorithm to another.

Following the investigation and the presentation of the evidence, it should be concluded that the minisatellites detected by *ab initio* algorithms tend to differs from one algorithm to the next.

It could be possible that divergences amongst these algorithms could be reduced by agreeing on a more detailed definition of what a minisatellite is. This definition should specify what parameters and their sensitivity level is, are allowed. As an example, on the ATRHunter's implementation of TR definition of TRF, there is a fourth parameter dedicated to terminal indels. This parameter does not exists in TRF's implementation, and is probably another cause of the difference between the two implementation results. In theory, techniques used by different algorithms should not result in different outcomes, when initially the goal is the same—finding minisatellites.

The software packages discussed in this chapter are yet to undergo usability evaluations, which is the topic for the next chapter, Chapter 6.

Chapter 6

Minisatellite detecting algorithms: usability comparison

6.1 Introduction

It was reported in Chapter 5 that different implementations of algorithms identified do not report on exactly the same minisatellites. This holds in the case of detection of a variety of PTRs as well as ATRs. It was explained how these differences came about in some cases, although a full understanding of the internal workings of the algorithms is required to fully explain the causes of these dissimilarities—something that is beyond the scope of this dissertation, not least because of the incomplete public information about these algorithms.

On the other hand, it should be borne in mind that discovering novel repeats is one of the reasons for employing an *ab initio* approach to minisatellite detection. In this sense, it might be advantageous that various algorithms detect different minisatellites since this might occasionally uncover interesting minisatellite-like strings that would have been missed if a rigid minisatellite definition has been used. Such minisatellites could be viewed as having more biological significance by one algorithm depending on what that algorithm regards as significant. In that case, it could be beneficial to allow the user to choose an algorithm or a combination of algorithms depending on the user's intentions.

A Graphical User Interface (GUI) could be created for the user to search for minisatellites. The user can set the parameters, and in addition to that, choose which algorithm(s) to use. The output of this search would show minisatellites detected by say, all the algorithms that were chosen, followed by minisatellites detected by only a single algorithm.

The problem with this solution however, is the different parameters used by these algorithms. The user will be expected to set the parameters for each algorithm chosen. This would mean that the user has to have a deeper understanding of every algorithm the user wishes to use. In addition, the user has to be able to tune the parameters between algorithms so that the algorithms detect desired results, i.e. constraining one algorithm's parameters, whilst relaxing the other, may not yield desired results.

Due to the restrictions on availability of source codes of some of these algorithms, and the time constraint placed for the completion of this study, this dissertation will not try to implement the above solution. This dissertation will however evaluate and improve a readily available algorithm developed by De Ridder et al [17] called FireμSat. The algorithm was initially developed to detect microsatellites. FireμSat is currently being extended as *'FireSat'* to detect minisatellites as well. For the purpose of this dissertation, the algorithm behind FireSat need not be discussed. The interested reader may consult De Ridder et al [18]. This dissertation uses the name FireμSatPlus to refer to the GUI behind FireSat.

In their study on improvements of OSS usability, Raza et at [67] suggested the following key factors:

- `User requirements:` understanding the user requirements helps in designing systems that are more usable.

- `Usability expert's opinions:` usability could be improved by involving experts during the design of the system.

- `An incremental design approach:` as opposed to building the whole system at once, different features in this approach are added incrementally. This means essential features are added first, whilst the nice-to-have features are implemented last.

- `Usability testing:` testing the system with real users will identify those requirements which are important to the user but were not apparent to experts.

- `Knowledge of user-centred design methods:` knowledge of these methods is likely to encourage designers to apply them appropriately.

With these key points in mind, this chapter will evaluate the available GUIs—FireμSat, TRF and Phobos—using Nielsen's ten heuristics in Section 6.2 as the expert evaluation technique. This is followed by task analyses to identify user requirements of the GUI in Section 6.3. Section 6.4 presents the improved GUI and the actual usability testing is reported on in Section 6.5.

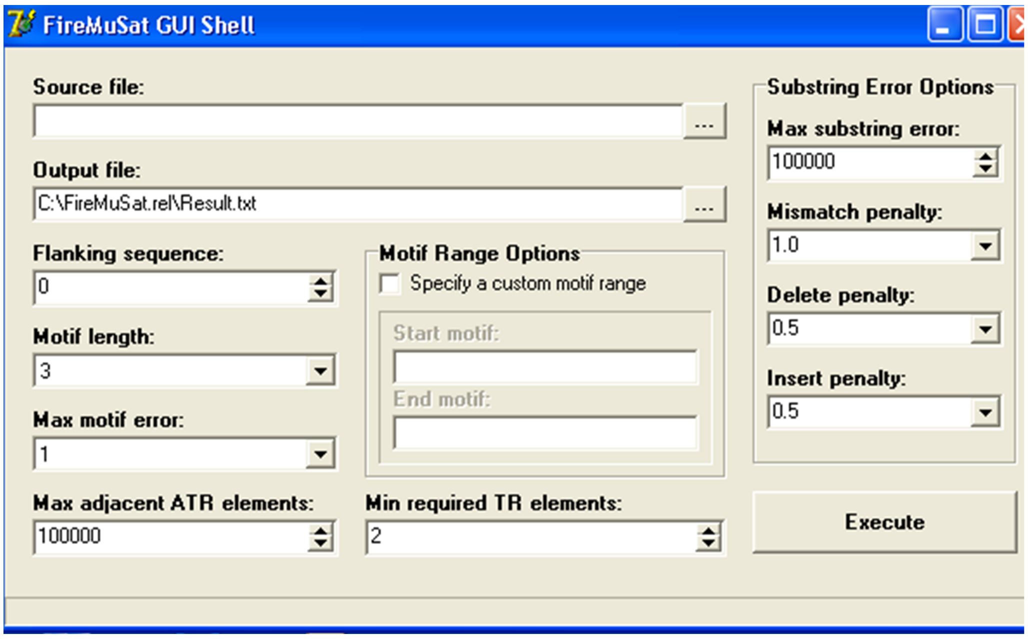
Table 6.1: Nielsen's ten heuristics.

Heuristics	**Description**
Visibility of system status.	User should be kept informed about what is going on, using appropriate feedback.
Match between system and the real world.	System should speak user's language, rather than system-oriented terms.
User control and freedom.	Users often choose system functions by mistake, in that case, they should be allowed to leave unwanted state without too much difficulty.
Consistency and standards.	Users should not wonder if actions, words, or phrases mean the same thing in different context.
Error prevention.	The system should make it hard to make errors.
Recognition rather than recall.	Objects, actions and options should be visible. Users should not have to remember information from one part of the dialogue to another.
Flexibility and efficiency of use.	The user interface should allow users to customise it so that frequent actions are easy to perform as the user wants. This can be accomplish by for example,using accelerator keys and keyboard short-cuts and allow users to create their own short-cuts. Accelerator may speeding actions for expert users.
Aesthetic and minimalist design.	Dialogues should not have irrelevant information.
Help users recognise, diagnose and recover from error.	Provide informative error messages in plain language.
Help and documentation.	Provide instructive help and documentation on the use of the system.

6.2 Heuristic evaluation of TR software GUI

It was mentioned in Section 6.1, that the opinion of a usability expert could significantly contribute to improving OSS usability. This would also applicable to Academic/Research Software (ARS). Table 6.1 shows ten heuristics proposed by Nielsen [56] as a guide when designing the usability aspects of a User Interface (UI). A heuristic is a guideline or general principle than can guide or critique a design decision [19]. Heuristics are generally useful for evaluating a UI design early during its development by software usability experts. The advantage of using this expert evaluation technique is that it is conducted early during system design, and therefore design flaws can be discovered and corrected earlier. The time and cost of re-performing usability testing with intended real users is thus minimised. After all, Lewis and Rieman [46] acknowledge that a user's time is almost never a free or unlimited resource.

A screen-shot of FireμSat, TRF and Phobos GUIs appears in Figures 6.1, 6.2 and 6.3 respectively. Typical user tasks were performed on the software packages and reported on the GUI based on the heuristics. Using Nielsen's ten heuristics, the three GUIs—FireμSat, TRF and Phobos—were evaluated and the summary of that evaluation is presented next.


FireMuSat GUI Shell

Source file:

Output file: C:\FireMuSat.rel\Result.txt

Flanking sequence: 0

Motif length: 3

Max motif error: 1

Max adjacent ATR elements: 100000

Motif Range Options
Specify a custom motif range
Start motif:
End motif:

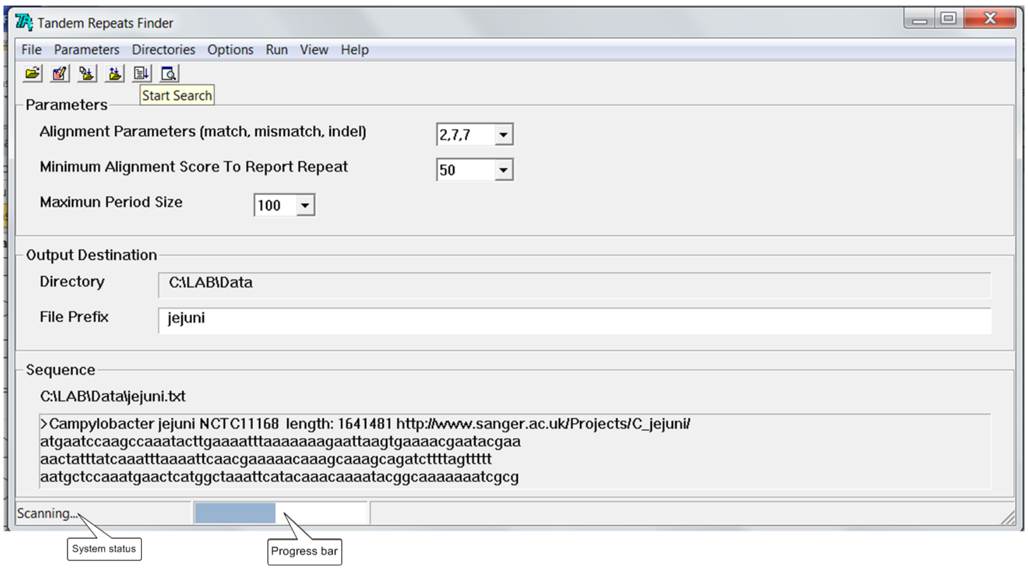
Min required TR elements: 2

Substring Error Options
Max substring error: 100000
Mismatch penalty: 1.0
Delete penalty: 0.5
Insert penalty: 0.5

Figure 6.1: FireμSat GUI.

Below is the evaluation of the three GUIs, i.e. TRF, Phobos and FireμSat, based on Nielsen's heuristics.

1. `Visibility of system status:`
   Requires that the user be informed about what the system is doing.



Parameters

Alignment Parameters (match, mismatch, indel): 2,7,7

Minimum Alignment Score To Report Repeat: 50

Maximun Period Size: 100

Output Destination

Directory: C:\LAB\Data

File Prefix: jejuni

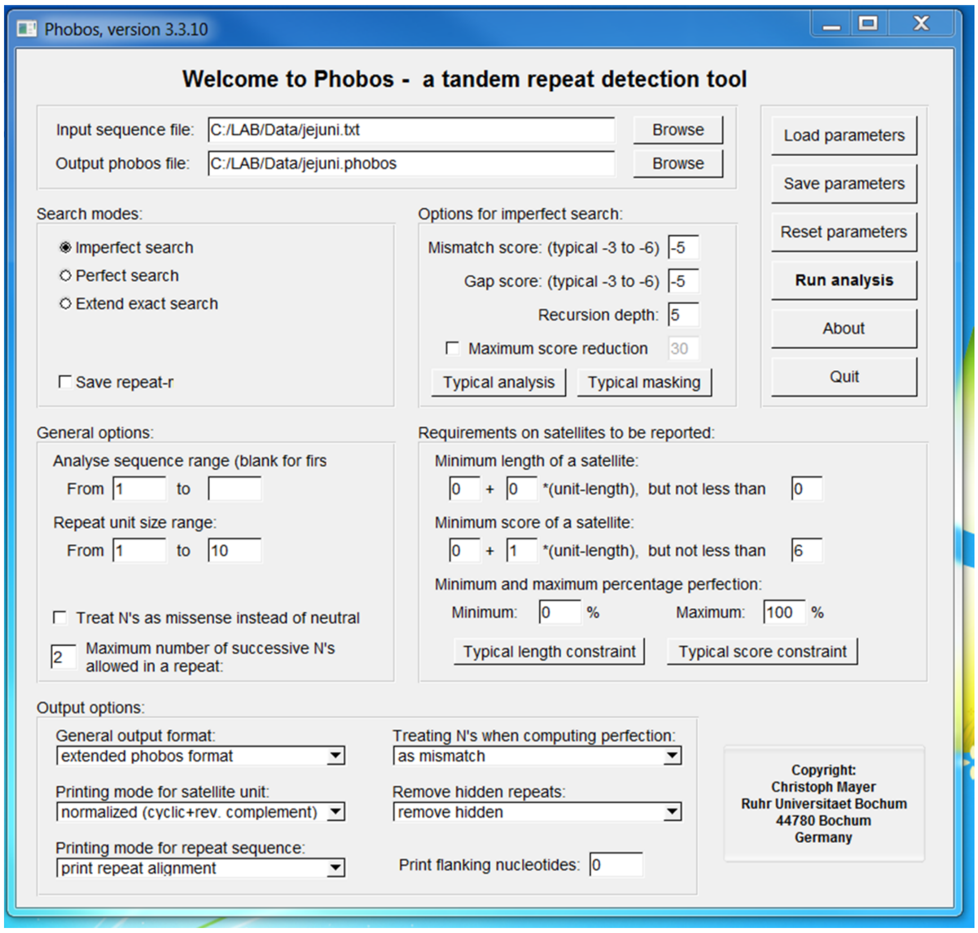
Sequence

C:\LAB\Data\jejuni.txt

```
>Campylobacter jejuni NCTC11168 length: 1641481 http://www.sanger.ac.uk/Projects/C_jejuni/
atgaatccaagccaaatacttgaaaatttaaaaaaagaattaagtgaaaacgaatacgaa
aactatttatcaaatttaaaattcaacgaaaaacaaagcaaagcagatcttttagttttt
aatgctccaaatgaactcatggctaaattcatacaaacaaaatacggcaaaaaaatcgcg
```

Scanning...

Figure 6.2: Tandem Repeat Finder GUI.



Welcome to Phobos - a tandem repeat detection tool

Input sequence file: C:/LAB/Data/jejuni.txt

Output phobos file: C:/LAB/Data/jejuni.phobos

Search modes:

- [✓ Imperfect search]
- Perfect search
- Extend exact search
- ☐ Save repeat-r

Options for imperfect search:

Mismatch score: (typical -3 to -6) -5

Gap score: (typical -3 to -6) -5

Recursion depth: 5

☐ Maximum score reduction 30

General options:

Analyse sequence range (blank for firs

From 1 to

Repeat unit size range:

From 1 to 10

☐ Treat N's as missense instead of neutral

2 Maximum number of successive N's allowed in a repeat:

Requirements on satellites to be reported:

Minimum length of a satellite:

0 + 0 *(unit-length), but not less than 0

Minimum score of a satellite:

0 + 1 *(unit-length), but not less than 6

Minimum and maximum percentage perfection:

Minimum: 0 % Maximum: 100 %

Output options:

General output format: extended phobos format

Printing mode for satellite unit: normalized (cyclic+rev. complement)

Printing mode for repeat sequence: print repeat alignment

Treating N's when computing perfection: as mismatch

Remove hidden repeats: remove hidden

Print flanking nucleotides: 0

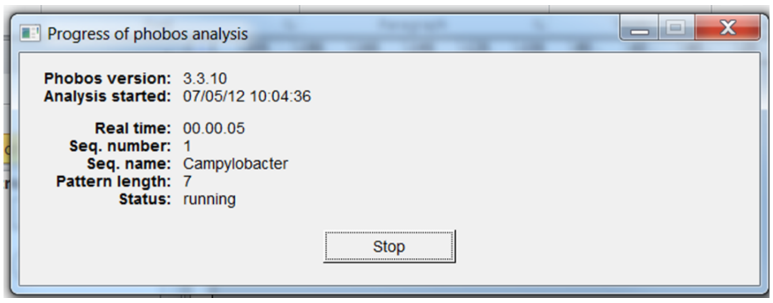
Copyright:
Christoph Mayer
Ruhr Universitaet Bochum
44780 Bochum
Germany

Figure 6.3: Phobos GUI.

This heuristic requires that users be informed about what the system is doing. As an example, searching a very large genome sequence on slower machines may require more time for an algorithm to complete executing. In cases like that it is important that the user be able to tell if the system is running or not.

- `TRF`: As indicated in Figure 6.2, the system status is indicated on the status bar on the bottom left of the GUI in the form of informative text and a progress bar. The progress bar gives an estimate indication of how far the search is from completion.
- `Phobos`: Phobos uses the status dialogue as shown in Figure 6.4. Although this dialogue indicates how long the search process has been running, it does not give clues about the progress towards completion.
- `Fire`μ`Sat`: No explicit system status is given.

The system status in both TRF and Phobos is made visible to users. Phobos's system status gives more information on the current process but fails to indicate the progress of the search. In other words, users cannot make an informed decision when requiring to interrupt the


Progress of phobos analysis

Phobos version: 3.3.10
Analysis started: 07/05/12 10:04:36

Real time: 00.00.05
Seq. number: 1
Seq. name: Campylobacter
Pattern length: 7
Status: running

Figure 6.4: Phobos search status indicating the systems busy status.

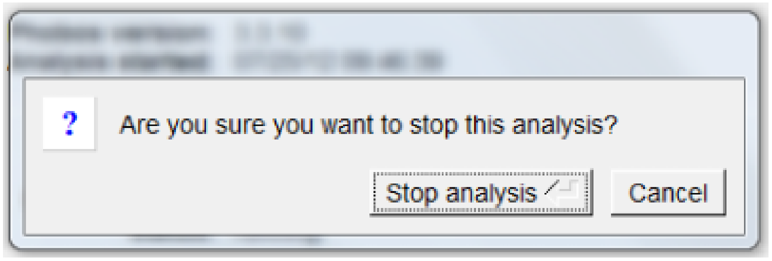

Are you sure you want to stop this analysis?

Figure 6.5: Phobos requiring a confirmation before executing a milestone action.

search. It might not be wise to interrupt a process that has been running for several hours and is near its completion. On the other hand, it might be a good decision to stop a process that has run for few hours and still requires few hours or days to complete. However, this feature may not be necessary for a process that takes a few seconds to complete.

2. `Match between system and the real world:`
   The choice of language used on the system should be consistent with the language the users speak everyday. Using natural everyday language and matching the system with the real world, allows the users to interact with a system in a way that seems natural to the users. In other words, the behaviour of the system as a result of some action should be predictable to the user. The following can be noted about the GUIs with regard to this heuristic:

   - `TRF:` The use of a magnifying glass icon on a piece of paper in Figure 6.2 may be misinterpreted for a search icon instead of view icon. However, TRF uses tooltips[*], which helps in guiding user's choices.
   - `Phobos:` The language used by Phobos (Figure 6.3) is in everyday language. In other words, computer jargon has been kept to the minimum.
   - `Fire`μ`Sat:` In Figure 6.1, the use of the word *Execute* which is intended to mean start searching, may be debatable by different users. Other words that could be used include *Start*, *Search* and *Run*.

   The choice of icons is very important as to not bring confusion on their meaning. Cultural differences may also affect their meaning, that is to say, an icon may mean one thing in one community and another thing in the other. In such cases, the use of tooltips can help clarify the misunderstandings.

3. `User control and freedom:`
   It is important for users to have control over the system. As an example, if wrong parameters were chosen and the user realises that just

[*]A tooltip is a text that appears when a mouse pointer is hovered over a GUI element without clicking on it.

after starting the search process, the user should not have to wait until a task is completed. As said before, with large genomes, it may take a while before a search task is completed.

- `TRF:` There is no explicit means of stopping the process once it has started.
- `Phobos:` While the search process is busy running, the status dialogue allows users to terminate the search by pressing the *stop* button anytime during the search as it can be seen on Figure 6.4.
- `Fire`μ`Sat:` The GUI (Figure 6.1), like the GUI of TRF, does not give this '*emergency exit*' control.

Phobos users are given control and freedom over the system by allowing the user to stop the search anytime. However, such freedom is restricted to the user by not providing information that may be necessary for good decision making.

4. `Consistency and standards:`
Section 3.3 of Chapter 3 brought up the importance of learnability as a usability attribute. Hence, this heuristic may support the learnability of the GUI by users, because previous knowledge gained from using other GUIs can be applied to the new GUI. Learning the new GUI is thus made easier.

- `TRF:` TRF uses the word *Period* to mean *motif*.
- `Phobos:` Phobos uses the word *Unit* and *Gap* to mean *motif* and *indel* respectively.
- `Fire`μ`Sat:` This GUI uses the Min and Max in reference to various settings such as motif error, required TR elements, substring error, etc. On the other hand, it refers to *start motif* and *end motif* under the heading of motif range. It would be more consistent to speak in this context of *Min motif length* and *Max motif length* respectively.

  FireμSat's uses of the word *Execute* is also in contrast with terminology used by Phobos and TRF, both using *Run*.

It was mentioned in Chapter 1 Section 1.2, that the terminology used in a discipline may vary over different authors. Some of the inconsistencies mentioned above are due to lack of standards within the research community investigating minisatellite detecting algorithms. These inconsistencies cause difficulties in transferring knowledge learnt from one software package to the other.

5. `Error prevention:`
   The risk that actions may put the system into an error mode, should be minimised by making them hard to perform, especially by novice users.

   - `TRF:` TRF has few parameters in which the user has to select from a list of predefined options. Providing predefined values makes it hard, if not impossible, for the user to choose invalid parameters.
   - `Phobos:` The use of a ComboBox[†] for fields like *Gap scores* in which its value is limited to a range from -3 to -6, could be more appropriate than EditBox[‡]. This is more error preventing because the user will know the available options to chose from, and this minimises invalid inputs from being entered.
   - `Fire`μ`Sat:` The controls in FireμSat GUI have been selected to help minimise users making errors.

   Providing users with predefined options makes it less likely to make errors. This is the route taken by TRF and FireμSat. It is however, not always possible to go this route as pointed out in point 9 below.

6. `Recognition rather than recall:`
   This heuristic requires that the system should reduce users' memory load. In other words, users should not heavily rely on their memory but the system should assist users to recall whatever is required by the system. Although this heuristic seems to be more appropriate for applications involving multiple of screens/pages , the provision of means to browse for input and output data files by all the GUIs is noticeable.

7. `Flexibility and efficiency of use:`
   Providing accelerator keys[§] and keyboard short-cuts makes the system

[†]A ComboBox is a GUI control that displays options available to the user as a drop down list. The user can either edit or choose from the list.

[‡]An EditBox is a GUI text field in which the user can edit.

[§]Accelerator keys appear as underlined characters on the GUI. The user has to press the `Alt + key` to access them.

more efficient to use by frequent users. In addition, providing alternative ways to accomplish the task makes the system more flexible to cater for different users.

- `TRF`: There are two ways to manipulate parameters, that is directly, or via the menu, on the GUI.
- `Phobos`: Parameters could be manually entered on the GUI or be loaded and/or saved for later reuse.
- `Fire`μ`Sat`: This GUI does not provide for this heuristic.

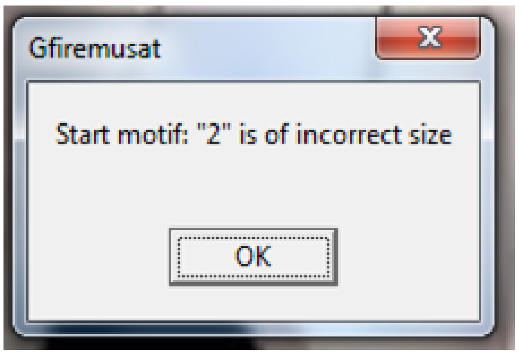
TRF and Phobos give alternative ways for manipulating parameters but they do not provide the use of short-cuts for speedy manipulation by expert users.

8. `Aesthetic and minimalist design:`
Too much unnecessary details on an application's GUI may hinder the user's performance by diverting the user's focus away from intended goals.

   - `TRF`: TRF shows the most minimalist design. The design looks simple and the flow—it is easier to see which parameter to set next—in setting parameters is predictable.
   - `Phobos`: Of the three GUIs, the Phobos GUI is the most detailed. Thus, the GUI may look complicated and overwhelming for novice users at first glance.
   - `Fire`μ`Sat`: FireμSat's GUI does not show inappropriate information that may interfere with the user's intentions. The GUI looks very minimal in its design.

   Complicated or aesthetically unpleasing GUIs may result in users avoiding to use them. However, as discussed in Chapter 3, usability has attributes which are subjective. Users also need to see how they are supposed to move around the GUI. Having to revisit parameters that have already been set as a result of unclear flow of control, may jeopardise productivity.

9. `Help users recognise, diagnose and recover from error:`
The GUI should not allow users to set the system into an error mode. If that cannot be prevented, then users should be able to realise the



Start motif: "2" is of incorrect size

Figure 6.6: FireμSat error message.

consequences of those actions and reverse or gracefully recover from such actions.

- `TRF`: It would be difficult, if not impossible for users to put the system into a fatal error state. This is because of preventative built-in safeguards taken by predefining only valid user inputs as available user options.
- `Phobos`: Figure 6.5 shows the results of an intentional, but possibly erroneous interruption of the search process. The latter gives the user an opportunity to recover from the possibly erroneous interruption.
- `Fire`μ`Sat`: Similarly to TRF, putting FireμSat in to a dangerous error state would be hard but not impossible. As an example, Figure 6.6 shows an error message that is displayed after the user has entered '2' as the starting motif. Although the message may be helpful to the user in recognising what went wrong, it does not give a hint on what the starting motif should be.

Although forgetting the input file may be regarded as an example of an error that users should be able to recognise from the error message and take required action, such errors are not actually fatal. Hence, such errors do not impose a significant usability threat.

10. `Help and documentation:`
    The availability of online documentation enhances the learnability of the system by providing user assistance online—users do not have to move away from the application GUI to find help.

- `TRF`: TRF provides help via context sensitive tooltips as shown in Figure 6.2 and documentation that can be directly accessed from the GUI.
- `Phobos`: No help is directly available from the GUI.
- `Fire`μ`Sat`: Similarly to Phobos, no help is provided.

Even though it is recommended that help facilities should be made available, it should not be seen as compensating for a poor design.

Table 6.2: GUI evaluation summary

Heuristic	**TRF**	**Phobos**	**FireμSat**
Visibility of system	✓	✓	—
Match between system and the real world	—	✓	—
User control and freedom.	—	✓	—
Consistency and standards	—	—	—
Error prevention.	✓	—	✓
Recognition rather than recall.	✓	✓	✓
Flexibility and efficiency of use.	✓	✓	—
Aesthetic and minimalist design.	✓	—	✓
Help users recognise, diagnose and recover from error.	—	✓[a]	—
Help and documentation.	✓	—	—

[a]This observation is solely based on accidentally stopping the search process prematurely. In this case only Phobos allows that and hence, gives users the opportunity to reconsider their actions.

This section highlighted some of the important aspects of usability as guided by heuristics. Table 6.2 summarises these aspects and gives an indication of which aspects were achieved , and which were overlooked. FireμSat seems to have overlooked the usability of its GUI more than the other GUIs.

It should be noted that a GUI's non-compliance with one or more of the above heuristics does not necessarily imply that the GUI has usability problems. It merely indicates a *potential* usability problem in relation to the heuristic concerned. In the commercial world, consideration of the extent to which a GUI matches heuristics such as these, is generally carried out by a team of experts within the company. However, in the ARS context where software development resources are usually limited, such an approach is not feasible. Hence, the above evaluation was conducted solely by the author of this dissertation.

Depending on intended users and their tasks, one heuristic may be more appropriate for one GUI than the other. Using the above evaluation, this dissertation will eventually suggests and implement some improvements on the FirμSat GUI. In doing that, this dissertation will follow the guidelines used in traditional software development for designing user interfaces. Therefore, task analysis is presented in Section 6.3 followed by requirements capturing in Section 6.3.2.

6.3 Task analysis

Lewis and Rieman [46] have pointed out the importance of understanding users and their tasks in order to design good interfaces. In other words, one should know the intended users of the computer system and what they will want to do with the system in order to know how the interface should be to best support users needs. Task analysis is the process of analysing the way people perform their jobs—that is, things they do, things they act on and things they need to know [19]. Richardson et al [69] noted that task analysis proved a method, to formally or semi-formally, analyse users' tasks. Analysing users' tasks may reveal other sub-tasks and the order they ought to be performed. Dix et al [19] mentions three techniques used in task analysis, and these are:

- Task decomposing: This looks at the way a task is split into sub-tasks, and the order in which these are performed.
- Knowledge-based techniques: These techniques focus on what the user already knows about the objects and actions involved in a task.
- Entity-relation based analysis: Thus is an object-based approach where the emphasis is on identifying actors and objects, the relationship between them, and the actions they perform.

After looking at these three techniques, it was decided that task decomposition would be a more appropriate technique to use within the context of this dissertation.

6.3.1 Task analysis of FireμSatPlus

Consider the following task that a typical user might like to perform using the FireμSatPlus GUI to search for Tandem Repeats (TRs):

1. `Task 1:` Search for approximate TRs of size n, where $2 \leq n \leq 100$.
2. `Task 2:` Search for perfect minisatellites of size range from y to z, where $y \leq z$ and $y, z = 2 \cdots 100$.

3. `Task 3:` Search as in Task 2, but starting at index position i of the DNA sequence. In addition, manipulate the nature of TRs to be reported by changing the following:

    a) Minimum number of TR elements to be reported.

    b) Maximum number of errors allowed.

    c) Penalties for mismatch and indel errors respectively.

    d) Maximum number of adjacent ATR elements.

The above is just a summary of tasks a user is likely to want to perform with the system. Each of the above tasks can be broken down into smaller sub-tasks that should be completed in order to reach the goals of the main task. These sub-tasks may need to be completed in a specific order. Dix et al [19] refers to a *Hierarchical Task Analysis*, as a hierarchy of tasks and sub-tasks and also plans describing the order in which this tasks should be performed.

Task 1: Decomposition

0. Approximate TRs of size n.
    1. Select size of n
    2. Select TR type.
    3. Start the search.

Plan 0: do 1 and 2 in any order
then do 3.

Figure 6.7: Task decomposition of task 1.

Figure 6.7 shows the sub-tasks that need to be completed for Task 1 to be realised. Within these sub-tasks, sub-tasks 1 and 2 can be completed in any order, whilst sub-tasks 3 should be performed last.

Similarly, Figure 6.8 and 6.9 denote sub-tasks to be completed in order for Task 2 and Task 3 to be realised respectively. In both situations, the sub-sub-tasks can also be completed in any order within a sub-task e.g. Task 2 has sub-task 1, with sub-sub-tasks 1.1 and 1.2, which can be completed in any order. In addition, a sub-task does not need to be fully complete before another sub-task is started. Take Task 3 for example, a user may start with sub-task 1 and complete sub-sub-task 1.1, then decide to do sub-task 2 before going back to sub-sub-task 1.2. In short, the first 7 sub-tasks of Task 3 can be performed in any order but they all have to complete before the last sub-task is initiated.

The importance of breaking down these main tasks into sub-tasks is to assist in capturing the requirements of the system and performing a detailed design of the interface.

6.3.2 Requirement capturing and detail interface design

Using the output of a task analysis as discussed in Section 6.3.1 the requirements of the GUI may be gathered and used to design the initial system. The breaking down of tasks into sub-tasks provides clues on what the system must provide to meet user's needs. The following requirements were identified as the results of the task analysis done above:

1. `Motif size`: Users should be able to set the motif size. The size could either be for a single motif or a range of motifs.

2. `TR type`: The TR type should be specified as either ATR and/or PTR. It should be possible to search for all TR types.

3. `Search index`: The index of the DNA sequence could be provided from at which the search is to start and/or end.

4. `Penalties`: The GUI should allow the user to set penalty scores for mismatches and indels.

Task 2: Decomposition

0. Perfect minisatellite of size range y to z.
   1. Select the motif size.
      1.1 Select size of y.
      1.2 Select size of z.
   2. Select TR type.
   3. Start the search
   Plan 0: do 1 and 2 in any order
   then do 3.

Figure 6.8: Task decomposition of task 2.

Task 3: Decomposition

```
0. Perfect minisatellite of size range $y$ to $z$.
   1.       Select the motif size.
            1.1       Select size of $y$.
            1.2       Select size of $z$.
   2.       Select TR type.
   3.       Select DNA search index
            3.1       Select start index position.
            3.2       Select end index position.
   4.       Set penalties
            4.1       Select mismatch.
            4.2       Select deletion.
            4.3       Select insertion.
   5.       Set maximum error
            5.1       Select motif error.
            5.2       Select substring error.
   6.       Set minimum TR elements
   7.       Set maximum adjacent ATR
   8.       Start the search
   Plan 0: do 1 to 7 in any order
.           then do 8.
```

Figure 6.9: Task decomposition of task 3.

5. `Maximum Error`: Users should be able to indicate the maximum number of errors that are to be allowed in a minisatellite. These errors could be motif and/or sub-string errors.

6. `Minimum TR elements`: Users should be able to specify the minimum number of Tandem Repeat Elements (TREs) before the minisatellite is reported.

7. `Adjacent ATR element`: There may be a need to specify the maximum number of ATR elements adjacent to each other.

The above requirements are converted into a design in conjunction with Nielsen heuristics. The requirements will inform the design on what parameters should be included in the system, whilst the heuristics help inform how these parameters should be provided, as seen in the next section.

6.4 Suggested FireμSat GUI improvements

In the light of the above interface requirements and the evaluation based on Nielsen's heuristics, this section attempts to improve FireμSat GUI. According to Dix et al [19], in UIs, dialogue is often taken to mean the order and structure of inputs and outputs between human and computer system. In HCI, there are two dialogue description notations used, and these are diagrammatic and textual notations. Examples of diagrammatic notations are: State Transition Networks (STN), Petri nets, state charts, etc. Textual notation includes grammars, production rules and event algebras to name a few. Dialogues notations make it easier to analyse the structure of the dialogue (or the interaction) separate from the actual program semantics. In that way, potential usability problems can be discovered. This dissertation will use the diagrammatic notation, which is widely used in dialogue design [19].

The State Transition Network (STN)—a type of formal specification [12] of part of system—in Figure 6.10 shows the proposed GUI design. According to Dix et al [19], STN helps to ensure that a design is complete and does not lead to error states from which it may be impossible to recover.

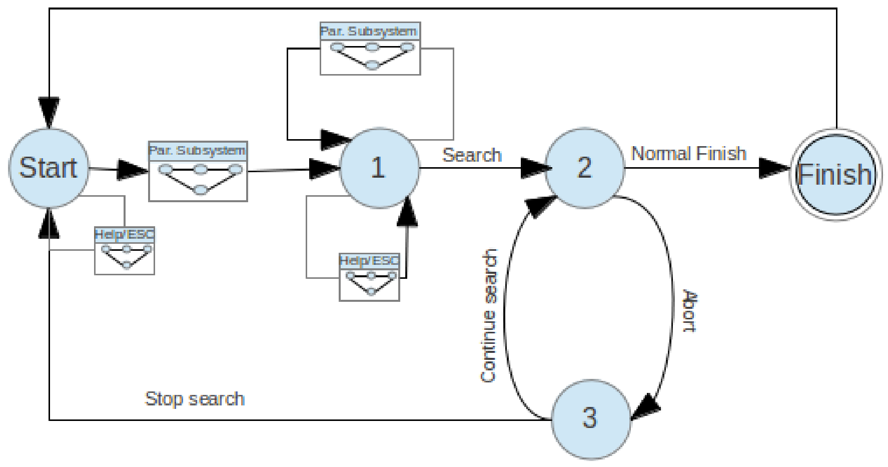


Figure 6.10: Complete STN for FireμSatPlus GUI.

The sequence of user interactions with the GUI is depicted in Figure 6.10. The circles in the figure represent the states in which the system can be at a given time during the interaction and the arrows denote the transitions which are triggered by users or system actions as labelled on the arrows. Other sub-systems that the system can get into are indicated as small rectangles. The user's interaction with this system can be viewed as follows:

- `Start state`: At this state a user may choose to quit without doing anything, or use the *help subsystem* which should provide assistance on how to use the GUI. Users familiar with the GUI can select parameters as required, which will lead to state 1.
- `State 1`: A user has an option to set or reset parameters before moving to the next state. *Help* and *ESC*[¶] options are also available immediately to the user at this state.
- `State 2`: If everything in the previous state is correct, that is, all required parameters have been set, the searching can begin on state 2. Unlike the former states, *help* and *ESC* on this state are not instantly available. Stopping the search at this state is considered critical and therefore requires precautions be taken before premature exit. This is because the system is busy searching and the results written on the output file may not be complete and therefore should not be used.
- `State 3`: Should the user opt to stop the search at state 2, the system may not allow this immediately, but should issue a warning and the opportunity to reconsider that action. This leads the system to state 3. If the user confirms that those actions are intentional, the search will stop (and go to state 1), otherwise the search continues (the system returns to state 2). This state is also critical, and hence no exit is possible.
- `Finish state`: If state 2 progresses to completion, then the system moves to the finish state which in turn goes back to the start state. The transition from state 2 to finish state is triggered by the system, and not the user as in other states.

6.4.1 Analysis of the STN

In analysing the dialogue design, Dix et al [19] mentions five properties of dialogue, and they are:

- Completeness: This property requires that the system remains intact during unforeseen circumstances as a result of user action. These actions should not have disastrous consequences.

[¶]ESC represents the escape system which is used as an exit.

- Determinism: Determinism requires that no single action, leads the system to two different states. The user should be clear what action leads to which state.

- Reachability: Reachability ensures that every state is reachable. In other words, there should be at least one path to each and every state.

- Reversability: This is a special case of reachability, which can be seen as the *undo*.

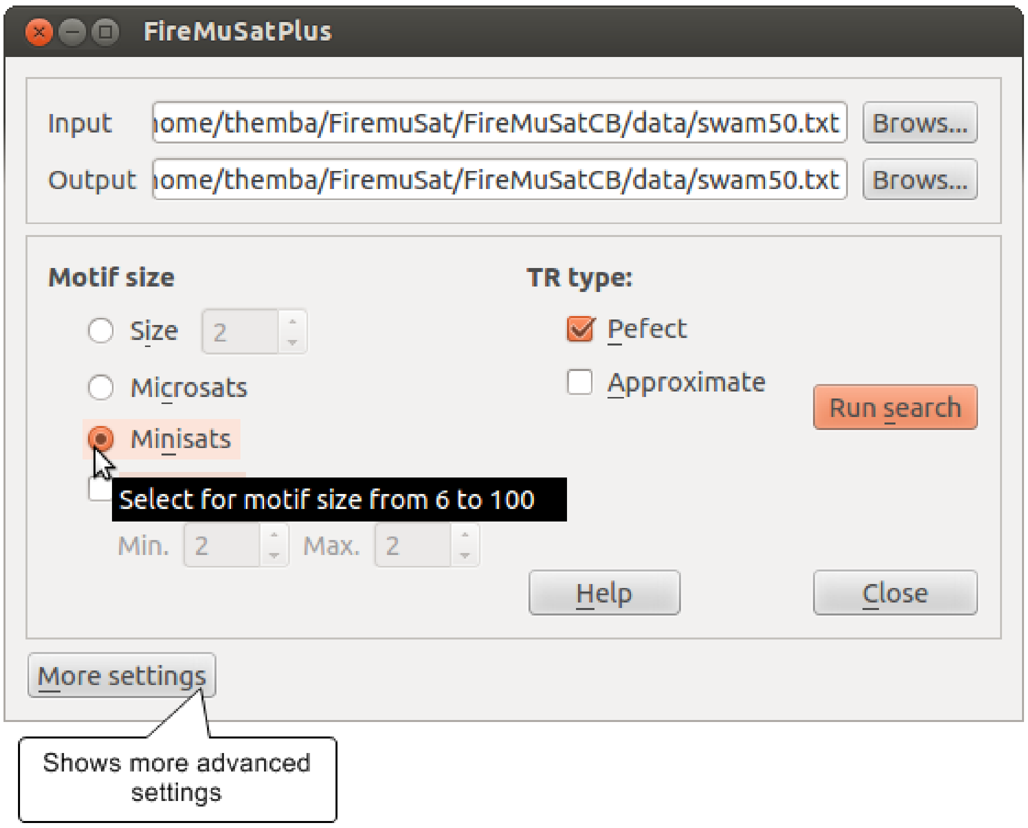
All these properties can be automatically checked on the dialogue. Looking back on Figure 6.10, it is observable that no unanticipated action by the user may lead the system into an unstable state. Each action leads to one state, and all states are reachable, whilst dangerous states are hard to reach. As an example, stopping the search in state 2 does not automatically go to the start state. Although, the start state is not the dangerous state, but to go to this state from state 2 is not considered as a desired route. This is ensured by placing state 3 as the intermediate state. The reverseability property can be seen in state 3 which gives the opportunity to undo an action. In summary , the presented STN is complete, deterministic, reachable and reverseable.

Using the parameter requirements identified in Section 6.3.2, the STN design, and Nielsen Heuristics as guidelines, a more detailed design of the GUI is constructed.

6.4.2 Detail design

This section uses the requirements captured during task analyses and integrates them with a created interaction dialogue, to produce a more detailed UI prototype. The prototype is constructed in line with Nielsen's heuristics and in an incremental fashion as suggested in Section 6.1, as follows:

Aesthetic and minimalist design: Aesthetic refers to the GUI's look and feel, whilst the minimalist design requires that the GUI does not contain irrelevant information. An aesthetic GUI should encourage users to use it, and minimalist design promotes productivity, by removing information that can distract users. A simple GUI also promotes learnability, enabling novice users to easily adapt to the GUI, and as they become familiar with the application, they can progress to use the application's more advanced functions.


FireMuSatPlus

Input: ome/themba/FiremuSat/FireMuSatCB/data/swam50.txt
Output: ome/themba/FiremuSat/FireMuSatCB/data/swam50.txt

Motif size: Size 2 / Microsats / [✓ Minisats]
Select for motif size from 6 to 100
Min. 2 Max. 2

TR type: [✓ Pefect] / Approximate

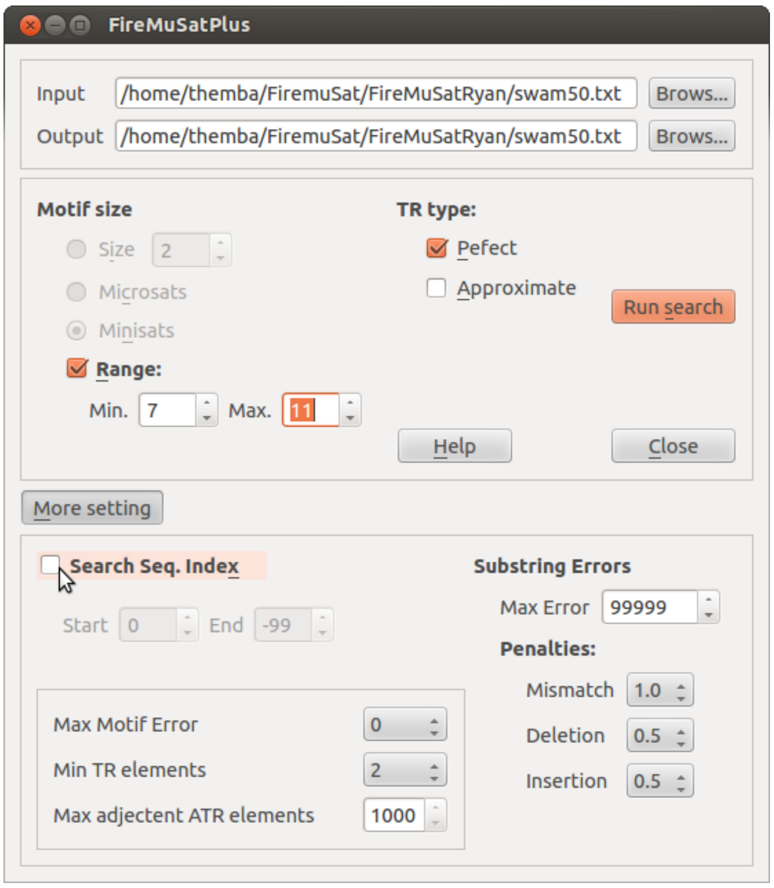
Shows more advanced settings

Figure 6.11: FireμSatPlus default run mode. This mode is suitable for both novice users as well as advanced users requiring a quick search. Starting the search without setting any parameter will give results of perfect minisatellites as these are the default settings. Pressing the '`More Settings`' button will reveal advance setting as shown in Figure 6.12.

Figure 6.11 shows a simple FireμSatPlus GUI prototype. This represents a minimalist design which is easy to use by non-frequent users. The design facilitates easy learning by:

- Using pre-set defaults: Defaults allow users to search for minisatellites without setting a single parameter. As users become familiar with the application, they can adjust the parameters as necessary. As an example, after inputting the data source and the output file, a user may just click on `Run search`. The results of this search will be perfect minisatellites, as indicated by default values in the figure.

- Use of tooltips: Context sensitive tooltips may assist new users to making choices of parameters, as displayed in Figure 6.11, by provid-



Input /home/themba/FiremuSat/FireMuSatRyan/swam50.txt
Output /home/themba/FiremuSat/FireMuSatRyan/swam50.txt

Motif size
Size 2
Microsats
Minisats
Range: Min. 7 Max. 11

TR type:
Pefect
Approximate

Search Seq. Index
Start 0 End -99

Max Motif Error 0
Min TR elements 2
Max adjectent ATR elements 1000

Substring Errors
Max Error 99999
Penalties:
Mismatch 1.0
Deletion 0.5
Insertion 0.5

Figure 6.12: FireμSatPlus extended mode. This mode is more suitable for advanced users knowing what they are doing. Selecting the '`Search seq. index`' will activate the '`Start`' value to 0 which is the start of the sequence and '`End`' value from `-99` to the size of the sequence.

ing additional information. In the same figure for example, the tooltip shows the motif size for minisatellites as from 6 to 100.

- Hiding advanced settings: Advanced settings, which may be more distracting for non-frequent users, are well hidden to such users , unless specifically requested. Users can access these settings by using the `more settings` button which will bring up an interface similar to that of Figure 6.12. These advance settings are for users interested in TRs which meet specific conditions. These conditions may be seen as too complicated for novice users.

On the other hand, Figure 6.12 displays advanced settings, which can be changed by a user who knows exactly how they want their results to appear. The screen is obtained by asking for '`more settings`'. These settings rely on default values and requires users who are knowledgeable about the application to change them.

Flexibility and efficiency of use: Frequent users are allowed to use accelerator keys, as shown in Figure 6.11 and 6.12. Not visible on the GUI are additional short-cuts which speedup productivity of more advanced users. Users can learn about these short-cuts in the documentation of the application. These key strokes are:

- `Ctrl + O` – open dialogue box for input file.
- `Ctrl + S` – open dialogue box for output file.
- `Alt + S` – start the search.
- `F1` – access help system.
- `ESC` – Exit the system.

In addition, users are allowed the opportunity of choosing parameters like the motif sizes, in an easy, flexible and efficient way—flexible in the sense that users have more than one way of making a selection.

The GUI is efficient in the sense that users can opt for quicker predefined options i.e. default options or individualised options.

Match between system and the real world: As was pointed out in Section 6.2, simple everyday language is encouraged over jargon-laden system language. The purpose of this software is to search for tandem repeats (TRs). Therefore it makes it more relevant to use the literal expression *'Run search'* to search for TRs, than using *'Execute'* which means initiating the algorithm that searches for TRs. The word *'Search'* hides the internal working of the software from the user and assures the user of the consequences of performing that action.

Consistency and standards: Consistency is important not only within the application itself, but amongst similar applications. In this regard, use of the word flanking sequence to mean the part of DNA sequence to be ignored during the search is not consistent with software packages like Mreps, TRF and ATRHunter. Instead of using "flanking sequence" terminology, *Search sequence index*, which basically refers to the sequence position to be analysed, is more comprehensible as everyday language, as shown in Figure 6.12.

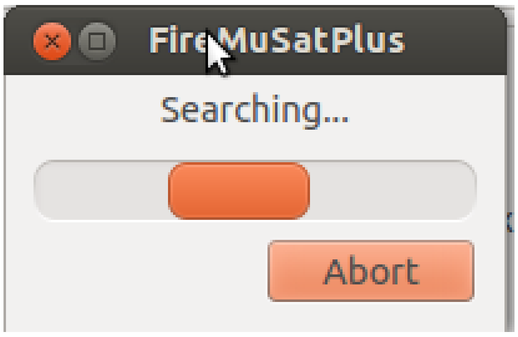


Figure 6.13: Initial FireμSatPlus system status and user control.

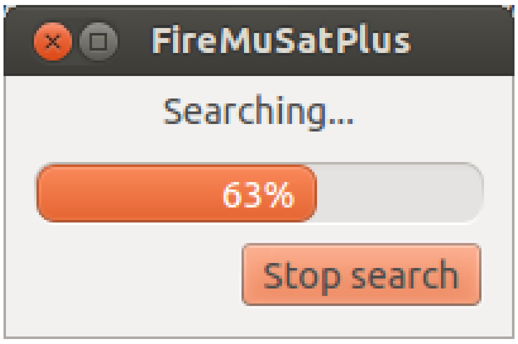
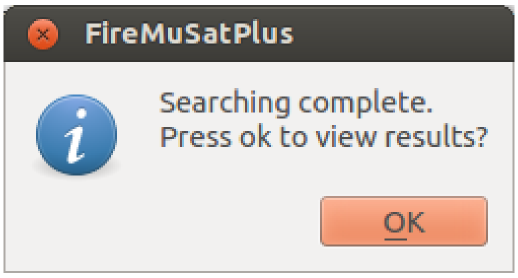


Figure 6.14: Newer improved FireμSatPlus running system status.

Visibility of system: Nielsen [56] has noted the importance of setting the system status visible to the users. The system status shows the user what the system is doing at a specific time. Users need to be assured that the system is still busy, or be informed if some error has occurred. Not indicating this to users may leave them thinking that the action they have



Searching complete.
Press ok to view results?

Figure 6.15: FireμSatPlus assuring the user of successful completion of the search.

performed is not being carried out. This may result in, for example, a button being pressed multiple times which may result in an unexpected system behaviour.

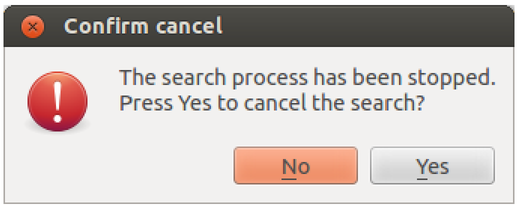
To avoid such confusion, the new FireμSatPlus GUI prototype displays its searching status as soon as searching commences. This can be seen on the GUI screen shot in Figure 6.13. During the re-evaluation of the design, it was apparent that this visibility is still limited to the user. This design only shows that the system is busy, but does not indicate to the user how far the progress is from completion. A more informative design with "progress bar" was adopted as indicated in Figure 6.14. Users are also assured if the search was successful as indicated on Figure 6.15.

User control and freedom: As was mentioned in Section 6.2, users need to be in control of the system, and not the other way around. With the older version of FireμSat GUI, not only did it leave the system status invisible to the users, it also did not give an obvious means of interrupting the search process before it is completed.

Figure 6.14 shows how this control is shifted to the user by allowing the user to stop the search before it completes. The design also gives enough information to the user to decide whether to stop or allow the search to continue to completion.

Error prevention and helping users recognise, diagnose and recover from error: Giving users wide control of the system may seem to be a good option, but it may introduce new problems when necessary precautions are not taken. Users make mistakes and, as Norman [58] acknowledges, since *to err is human* , measures need to be taken to prohibit such mistakes. As an example, the means that provides user control of the system may turn out to be disastrous if the '*Stop search*' button is mistakenly pressed. For small amounts of data it may not be such a problem, but with large amounts of data, where a user has to wait hours for results, accidentally pressing '*abort*' would mean that the whole search process has to be restarted.

The FireμSatPlus GUI prototype prevents such erroneous disruptions by informing the user about the consequences of that action, and requests a confirmation that the action was intentional. Figure 6.16 shows the systems dialogue box interventions as a result of such actions. As a result, if the action was not intentional, users may recover from it, by reacting appropriately. The warning message makes it harder for users to over-look this, by



The search process has been stopped.
Press Yes to cancel the search?

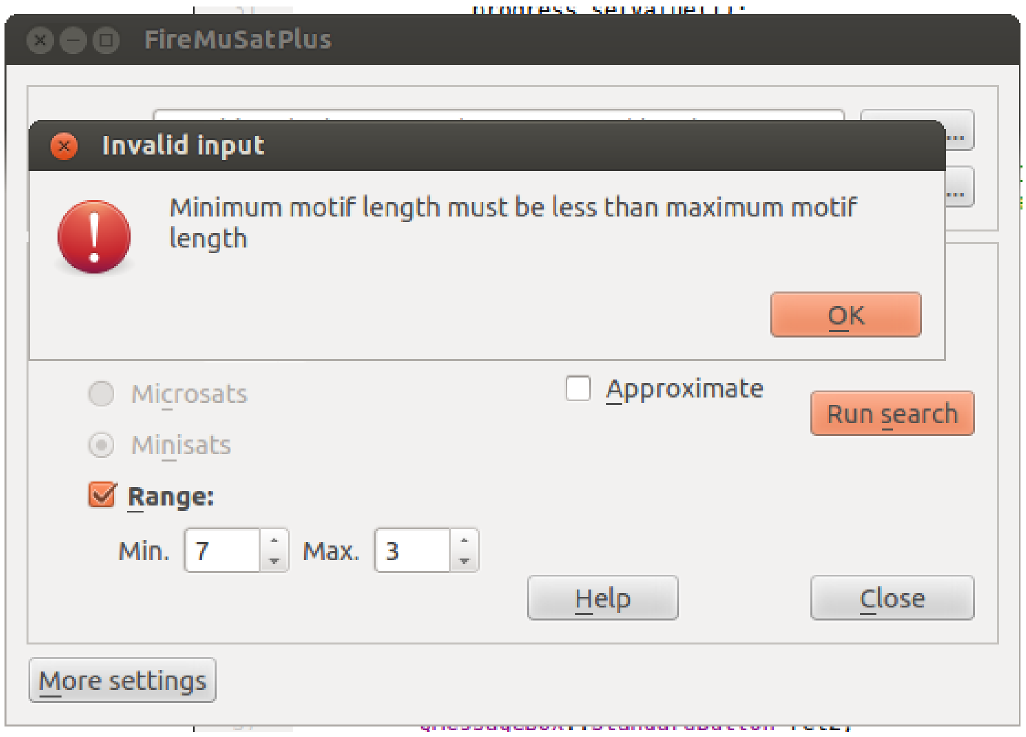
Figure 6.16: FireμSatPlus error prevention and recovery. *No* is the default button requiring the user to explicitly select *Yes* to confirm the cancellation, which further complicates unintentional abortion of the process.

requiring them to explicitly press the *Yes* button to confirm that cancelling the task was intentional.

The flexibility gained by permitting users to specify exactly the range of motif sizes they desire, has consequences. Users could enter infeasible ranges for the motif sizes. This can be prevented by enforcing a precondition that the minimum motif size should always be equal or less than the maximum motif size. Should this condition not be the case, an appropriate error message is displayed for the user to react. Such a message is shown in Figure 6.17.

Help and documentation: Tool tips, an example of which can be seen in Figure 6.11, are informative and can be used as a quick reference for non-frequent users. In some cases the users may require detailed information and extra form of assistance is required. The *help* facility is provided for that purpose. Although this facility is not currently implemented, it is envisaged that it should contain the application's printed manual and online tutorial to assist users on how to complete common tasks. In the case of an ARS product, the bug reporting facility should be implemented, to allow users to report and offer suggestions on improvements to the software as noted by Nichols and Twidale [55].

Using Nielsen's heuristics to evaluate the GUI has shown that many usability issues could be detected early during the design, and be addressed. Under normal ARS conditions, the GUI should be made available to the public for users to do real testing. However, for the sake of this dissertation, this GUI was subjected to potential intended users and their feedback is given in Section 6.5. The source code for this GUI is attached in appendix E for interested readers.



Minimum motif length must be less than maximum motif length

Figure 6.17: FireμSatPlus error message as a result of invalid parameters' combination.

6.5 Usability testing on FireμSat and FireμSatPlus GUIs

The minisatellites user community is currently very small. As a result, it is a challenge to find a truly representative sample for this usability testing. In the light of that, a small sample group of students in the School of Computing at University of South Africa (UNISA) were approached to mimic real users. Appendix A provides more detail pertaining to the conduct of usability testing. The group was given tasks to complete and a questionnaire to complete in order to evaluate their attitudes towards the two GUIs. Two groups of six users were used for this purpose.

Table 6.3: Likert scale scores for FireμSat and FireμSatPlus.

	Strongly agree		**Agree**		**Maybe**		**Disagree**		**Strongly disagree**	
Heuristics:	**FS**[*]	**FS+**[†]	**FS**	**FS+**	**FS**	**FS+**	**FS**	**FS+**	**FS**	**FS+**
Visibility of system	0	5	1	5	2	1	5	1	4	0
Match between system and the real world	0	0	1	7	4	5	6	0	1	0
User control and freedom	0	3	0	6	4	3	4	0	4	0
Consistency and standards	2	1	5	7	5	4	0	0	0	0
Error prevention	1	4	7	7	3	1	1	0	0	0
Recognition rather than recall	0	3	5	7	6	2	1	0	0	0
Flexibility and efficiency of use	0	6	0	6	1	0	6	0	5	0
Aesthetic and minimalist design	4	6	3	6	4	0	1	0	0	0
Help users recognise, diagnose and recover from error	2	8	4	3	2	1	3	0	1	0
Help and documentation	0	4	0	5	0	3	0	0	12	0
Total	**9**	**40**	**26**	**59**	**31**	**20**	**27**	**1**	**27**	**0**
Percentage total	**7.5%**	**33.3%**	**21.7%**	**49.2%**	**25.8%**	**16.7%**	**22.5%**	**0.8%**	**22.5%**	**0.0%**

[*]FS: FireμSat.

[†]FS+: FireμSatPlus.

Appendix C and D gives the tasks and questionnaire used respectively. The questionnaire was adapted from the questionnaire used during the study conducted by Ssemugabi and de Villiers [79]. The questionnaire uses a five point Likert scale as shown below,

Strongly Agree	5
Agree	4
Maybe	3
Disagree	2
Strongly Disagree	1

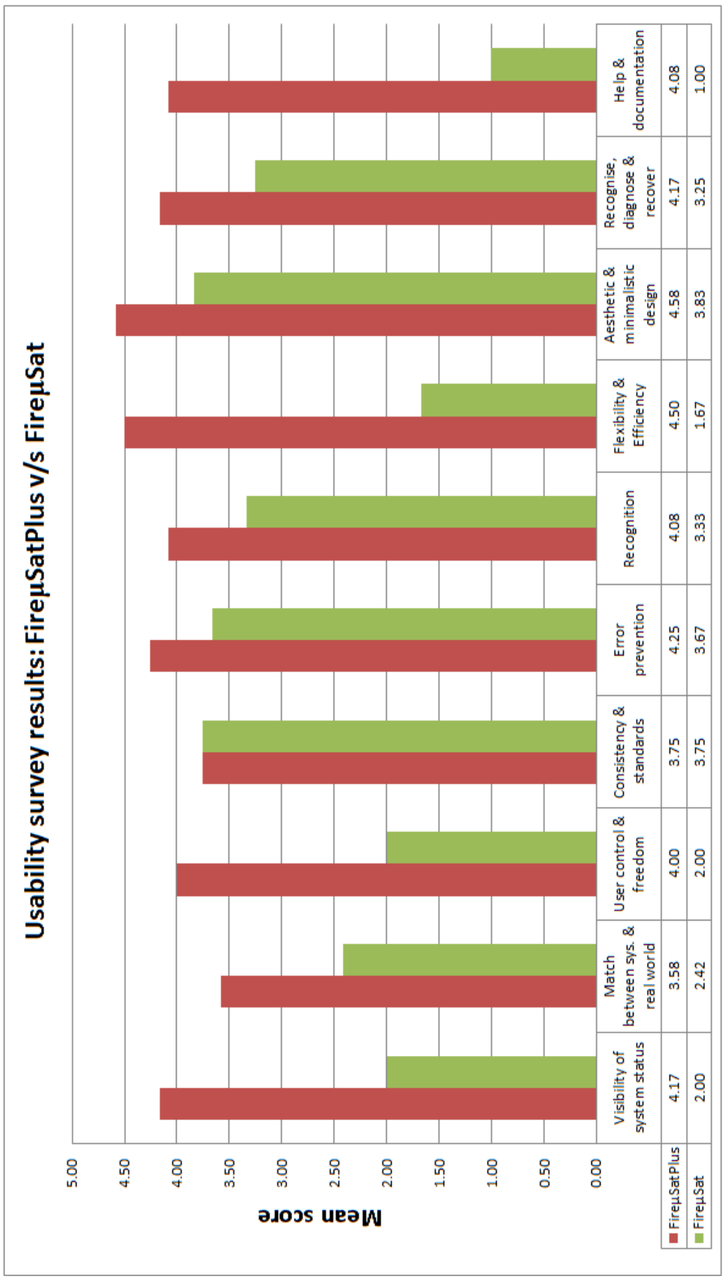
where users are required to select one of the five options for each question.

The results of the tests are summarised in Table 6.3. This table shows for every question, the total number of points awarded by the users. Figure 6.18 depicts the resulting mean scores of the two GUIs on each heuristic. From the figure, it can be noted FireμSatPlus prototype has an improved usability in comparison with the older version—FireμSat. This is especially noticeable in the following Nielsen heuristics:

- Visibility of system status. This is because the older version does not show the current system status.
- User control and freedom. The old version does not allow users to interrupt the system in any way.
- Flexibility and efficiency. Catering for both novice and expert users is not supported by the old version.
- Help and documentation. No form or indication of where assistance could be found is available on FireμSat.

The proposed new version has addressed the above problems by using suggested solutions to OSS usability issues as was shown in Section 6.1. Clearly, it is observable that there is an improvement in FireμSatPlus over FireμSat. However, to test if these differences are statically significant, a student's t-test[‖] that does not assume equal variance was performed.

[‖]Student's t-test is used to assess whether the means of two groups are statistically different from each other.



	Visibility of system status	Match between sys. & real world	User control & freedom	Consistency & standards	Error prevention	Recognition	Flexibility & Efficiency	Aesthetic & minimalistic design	Recognise, diagnose & recover	Help & documentation
🟥 FireµSatPlus	4.17	3.58	4.00	3.75	4.25	4.08	4.50	4.58	4.17	4.08
🟩 FireµSat	2.00	2.42	2.00	3.75	3.67	3.33	1.67	3.83	3.25	1.00

Figure 6.18: Usability testing mean scores.

The results of the test: The mean score of FireμSat ($Mean = 2.69$, $SD = 1.002$[**], $N = 10$[††].) is significantly smaller than the score for FireμSatPlus ($Mean = 4.11$, $SD = 0.302$, $N = 10$.) using the two-sample t-test for unequal variances, $p < 0.00124$. This low p value implies that the difference between FireμSatPlus and FireμSat is not by chance. Even though the underlying distribution is not normal, as required by the t-test, the results support the conclusion that can be drawn by simple observation: namely that in 90% of the questions, the average scores for FireμSatPlus were higher than FireμSat. In the remaining case (10% of the observations) the two GUIs were equally ranked.

Figure 6.19 and Figure 6.20 give a summary of users' perceptions about the usability of both FireμSat and FireμSatPlus. From the figures, it is clear that users ranked FireμSatPlus more favourably than FireμSat.

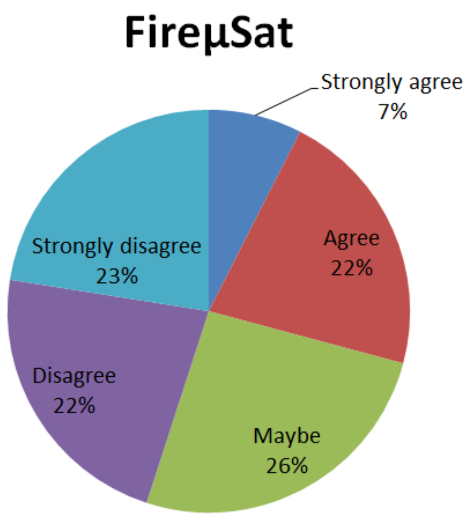


Figure 6.19: Summerised Likert scale scores for FireμSat.

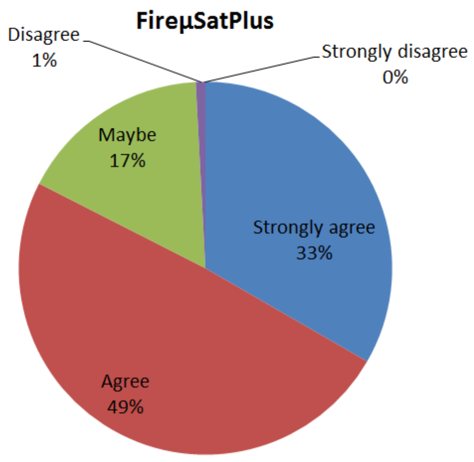


Figure 6.20: Summerised Likert scale scores for preferences for FireμSatPlus.

6.6 Conclusion

This chapter has attempted to identify usability issues regarding selected ARS applications using expert evaluation techniques. These techniques

[**]Standard deviation shows how much variation exists from the average.
[††]N is the number of observations

are important early during the design, as they can cut costs in software development. These costs include, but are not limited to, monetary costs; they are also the time and resources required when usability testing with real users is conducted.

Using the results of this evaluation and the suggested guidelines, a new version of FireμSat GUI was developed. This version was subjected to usability testing and the result showed an improvement over the old version.

Chapter 7

Conclusion

7.1 Introduction

This study has investigated two themes, namely, the consistency of the output produced by a selection of open source minisatellite detecting algorithms, and the usability of that software. Detailed discussion of the outcomes of these investigations was provided in Chapters 5 and 6 respectively.

This chapter outlines the journey taken by this study to investigate these two themes. Section 7.2 give a brief review of the investigations. The implications of the findings and their limitations are looked at in Section 7.3 and 7.4 respectively. The chapter ends with general conclusions about the whole dissertation.

7.2 Overview

The importance of detecting Tandem Repeats (TRs), and hence, minisatellites was discussed in Chapter 1 in the beginning of this dissertation. During that discussion later on, two approaches used by algorithms to detect minisatellites were noted: the *library or database* based approach, which is limited to searching for known motifs only; and the *ab initio* based approach, with the potential to discover novel motifs and hence, new minisatellites. The focus of this dissertation was only on the algorithms that used the latter approach to detect minisatellites.

Chapter 1 also highlighted the importance of using Academic Research

Software (ARS) and/or Open Source Software (OSS), especially in developing countries. It was said that these software could decrease costs and should therefore be considered as an alternative to a proprietary software (PS).

As the advantage of using *ab initio* based approach was made obvious—finding novel minisatellites without a priori information about motifs—the question of whether two algorithms would detect the same minisatellites if given the same genomic sequence, remained debatable. To investigate this question the following hypotheses were set out:

- H_1: Equivalent minisatellites (PTRs) are detected by different algorithms, when no mismatches and/or *indels* are allowed.
- H_2: Equivalent minisatellites (ATRs) are detected by different algorithms, when mismatches and/or *indels* are allowed.
- H_3: Non-equivalent minisatellites (PTRs and ATRs) are detected by different algorithms when the meaning of "approximate" assumed by the algorithms varies from one to the next.

Gathered evidence from the literature review discussed in Chapter 3 suggested that most OSSs have problems with regard to usability. It was suggested in the literature that some of these problems are as a result of the way in which OSS is developed. The similarities between OSS and ARS with regard to their development suggested that both these software would experience similar usability issues. Hence, the literature suggested that borrowing guidelines from PS development could potentially alleviate these problems. This dissertation has therefore investigated the usability of minisatellite detecting ARS packages that are based on an *ab initio* approach. In addition, this dissertation used the guidelines suggested by academic papers to improve one of the user interfaces. In summary, this theme was investigated by attempting to answer the following questions:

1. To what extent do the selected minisatellite detecting academic/research implementations follow usability guidelines suggested in the literature?
2. To what extent would implementing the usability guidelines as proposed in academic papers improve the usability of one of the ARS detecting minisatellites?

The overall objectives of this dissertation were:

- To learn whether and to what extents there are similarities and/or differences in output minisatellites detected by the algorithms implemented in the ARS software under study that rely on an *ab initio* approach.

- To assess the usability of open source minisatellite detection software and consider how it could be improved.

The findings of this study and whether or not its research questions were met are presented in Section 7.3. Section 7.3 also provides information on the assessment of whether the above objectives of this study were attained. That is followed by the limitations of this study in Section 7.4.

Chapter 4 introduced the instruments that were used in these investigations. Firstly, a set of experiments would be conducted to test the three hypotheses. Secondly, an evaluation by human expert and usability testing would be used to investigate the second theme. Chapter 4 also indicated how the data from these two investigations would be analysed. The actual investigation is reported in Chapter 5 and 6 for the investigation of objectives 1 and 2 respectively. The next section summarises the findings of this dissertation.

7.3 Implication of the findings

To remind the reader, this dissertation investigated two main questions. The first question that was investigated was:

> Do the selected ARS implementations behave consistently with respect to the occurrence and positioning of minisatellites, even though they do not rely on information provided *a priori* about the positioning of minisatellites?

In order to answer this first question, three hypotheses (H_1, H_2 and H_3) were formulated as mentioned in Section 7.2. After the appropriate experiments were conducted, it was seen that the obtained results did not support hypotheses H_1 and H_2 relating to equivalence of minisatellites detected. In the case H_2, the results showed very few similar minisatellites whilst most of them were different. In other words, there were more differences than similarities between reported minisatellites. Looking at these results may make one to wonder what the causes of these differences are . One possible explanation could lie in the definition of a TR used by a particular algorithm. Hypothesis H_3 investigated this possibility.

The assumption here was that these algorithms may be implemented based on the same or different definition of a TR—minisatellite. Initially, the different definitions of TR were used and later the same definition was used. The results from two cases were compared. The results obtained when the comparison based on same definition and different definition showed differences in output minisatellites , although the same input data was used. This led to the conclusion that *ab initio* approaches do not yield consistent minisatellites within different algorithms. In other words, what is reported as being a minisatellite by one algorithm, is not always reported by the rival algorithms. Although identifying novel repeats is desirable, and this is indeed a way of achieving this, it raises concerns in deciding which of these outcomes are truly minisatellites.

It is with no doubt that the results of the output minisatellites were not consistent in all the hypotheses. Potential causes of these differences were briefed in Chapter 5. It is however, proposed that a more formal definition of what constitutes a TR, hence a minisatellite, should be derived and agreed on. All these algorithms should adhere to the same definition, to minimise or eliminate those concerns entirely. This proposal is based on the observation that when using the same definition, more similar minisatellites were reported that when different definitions were used. This can be seen in Figure 5.7 as one looks at the total number of similar minisatellites reported by TRF and the variants of ATRHunter. With all of that being said, this study has met its first objective as mentioned in Section 7.2.

The second question addressed by this dissertation is:

> How usable are the software packages previously investigated to address the first question?

To aid the investigation, this question was decomposed into two sub-questions as outlined in Section 7.2. The first sub-question related to the usability of the packages per se. It was investigated using Nielsen's ten heuristics. Out of ten literature-derived heuristics, the package compliance varied between about 30% and 60%. It should be noted an interface was treated as complying with a heuristic, even if its compliance was only partial. Therefore, there range above could be slightly less than indicated. With that in mind, one should agree that thirty percent compliance is rather too low to say that guidelines were followed.

The second sub-question related to the effectiveness of usability improvements made to one of the packages. To answer the second sub-question, a survey was conducted. The results of the survey suggested that there was an overall of 34% improvement on the new prototype developed using the suggested guidelines. To be more precise , only 48% of the users agreed that the old interface was usable, whilst 84% of users were positive about the new interface.

To return to the main usability question, it is clear from the results obtained that one can confidently say that these packages are not user friendly. The investigation did not stop on saying the packages are hard to use, but also offered evidence that they can be improved. This outcome supports the second objective of the study.

7.4 Limitations of the study

This study only focused on a selection of ARS implementations of minisatellite detection software packages. Although PS alternatives for these packages may exist, this study did not investigate such alternatives. This dissertation also did not fully explain the precise reasons for differences across packages in reported minisatellites. It was however mentioned that the full explanation is beyond the scope of this study.

During usability evaluation of the ARS packages, only one user interface was upgraded. Under ideal circumstances, it would have been interesting to attempt a usability upgrade of more than one, and preferably of all ARS packages that were examined. This would have lent stronger support to the claims made in this study. While time limitations precluded such a comprehensive undertaking, the results to date offer prima facie evidence that there is considerable scope for improving the usability of the existing ARS packages.

Furthermore, this study focused only on subjective measures, guided by Nielsen's heuristics [56] . Apart from these measures, software usability testing could also be measured in terms of scenario completion success rates, adherence to dialogue scripts and error rates. Measuring these metrics in an ARS development environment would have required special data loggers to record each user's actions installed with the software to be tested. As was indicated in Chapter 3, the OSS community often uses tools to automate bug reporting. Using such tools to record users' actions on the interaction level so as to identify usability problems, could complete usability testing in such environments.

Clearly, the conclusion to this study applies only to the investigated packages. This fact, together with the aforementioned limitations affect the generalisability of this study and should be kept in mind when assessing the implications of this study.

7.5 Future research

The potential future research that arises from this study would be to investigate PS implementations that detect minisatellites, and to compare those findings with the findings of this study. The evidence collected from that investigation might suggest a more generalised conclusion that all *ab initio* based approaches do not produce identical results, and not just those implemented as ARS. This in turn would strengthen the call for standardised definitions of TRs—minisatellite.

Other potential research would be to evaluate ARS implementations using other usability metrics, such as scenario completion success rates, adherence to dialogue scripts, and error rates. It might be possible to identify reliable tools to automate this in an ARS environment.

Another point worth mentioning is with regard to principles borrowed from PS development practises. Evidence that the PS development practises work should be investigated by applying them and testing results, preferably over a sample of more than one package.

A methodology to integrate the two development disciplines represented by ARS development and PS development should be investigated to provide guidelines on how to integrate the two approaches without compromising one another.

7.6 Conclusion

This chapter gave a summary of the important aspects of this dissertation. It indicated what the problems were, and the significance of investigating those problems. The instruments used and procedures followed in investigating those problems were also pointed out, with reference to the relevant chapters which in turn give a detailed justification of the chosen methodology. The justification of such a methodology made it convincing that the methods used would undoubtedly answer the questions needed to address these problems.

After the investigation, conclusions were drawn based on the evidence collected. Section 7.3 provides the inference of those conclusions to this dissertation. The extent to which the conclusions of this study can be generalised was also considered, noting the limitations of this study. Possible future research was mentioned which could address some of these limitations.

To summarise, firstly, this study has concluded that there is inconsistency between the minisatellites detected using the *ab initio* approach by algorithms implemented as ARS packages. Secondly, this investigation confirmed that there are also usability problems with the software packages examined. It also offered evidence of how their usability could be improved in the future.

Bibliography

[1] Allauzen, C., Crochemore, M., and Raffinot, M. Factor oracle: A new structure for pattern matching. In SOFSEM 99: Theory and Practice of Informatics (1999), Springer, pp. 295–310. [cited at p. 36]

[2] Andreasen, M., Nielsen, H., Schrøder, S., and Stage, J. Usability in open source software development: Opinions and practice. Information technology and Control 35A, 3 (2006), 303–312. [cited at p. 50]

[3] Armour, J., Anttinen, T., May, C. A., Vega, E. E., Sajantila, A., Kidd, J. R., Kidd, K. K., Bertranpetit, J., and Paabo, Svante Jeffreys, A. J. Minisatellite diversity supports a recent African origin for modern humans. Nature Genetics Journal 13, 2 (1996), 154 – 160. [cited at p. 6]

[4] Author 1, Author 2 and Author 3. [cited at p. 21, 67]

[5] Bennett, S., Lucassen, A., Gough, S., Powell, E., Undlien, D., Pritchard, L., Merriman, M., Kawaguchi, Y., Dronsfield, M., Pociot, F., Nerup, J., Bouzekri, N., Cambon-Thomsen, A., Ronningen, K., Barnett, A., Bain, S., and Todd, J. Susceptibility to human type 1 diabetes at IDDM2 is determined by tandem repeat variation at the insulin gene minisatellite locus. Nature Genetics Journal 9, 3 (1995), 284–376. [cited at p. 6]

[6] Benson, G. How does tandem repeats finder work? Online: http://tandem.bu.edu/trf/trfdesc.html. accessed 01 May 2011. [cited at p. 29, 83]

[7] Benson, G. Tandem repeats finder. Online: http://tandem.bu.edu/trf/trf.html. accessed 20 January 2010. [cited at p. 29]

[8] Benson, G. Tandem repeats finder. Nucleic acids research 27, 2 (November 1999), 573–580. [cited at p. 20, 29]

[9] Bodker, M., Nielsen, L., and Orngreen, R. Enabling user centered design processes in open source communities. In Usability and Internationalisation, HCI and Culture, 2nd International Conference on Usability and Internationalisation, UI-HCII 2007. [cited at p. 50]

[10] Boeva, V., Fridman, M., and Makeev, V. Relationship between micro- and minisatellites in the human genome]. Biofizika 51, 4 (2006), 650. [cited at p. 16]

[11] Camp, N., Cofer, H., and Gomperts, R. High-throughput BLAST. Online:`http://www.sgi.com/industries/sciences/chembio/resources/papers/HTBlast/HT_Whitepaper.html`. accessed 25 June 2010. [cited at p. 17]

[12] Carroll, J. HCI models, theories, and frameworks : toward a multidisciplinary science. San Francisco, Calif.: Morgan Kaufmann, 2003. [cited at p. 107]

[13] Çetin, G., and Göktürk, M. Usability in open source community. ACM Interactions 14, 6 (2007), 38–40. [cited at p. 9, 10, 50]

[14] Çetin D. Verzulli, G., and Frings, S. An analysis of involvement of HCI experts in distributed software development: Practical issue. Online communities and social computing, 2nd international Conference, HCII 2007, pp. 32–40. [cited at p. 51]

[15] Danek, A., Pokrzywa, R., Makałowska, I., and Polański, A. Application of the burrows-wheeler transform for searching for approximate tandem repeats. In Pattern Recognition in Bioinformatics, vol. 7632. Springer Berlin Heidelberg, 2012, pp. 255–266. [cited at p. 20]

[16] De Ridder, C. Flexible finite automata-based algorithms for detecting microsatellites in dna. Master's thesis, University of Pretoria, 2010. [cited at p. 9]

[17] De Ridder, C., Kourie, D., and Watson, B. FireμSat: Meeting the challenge of detecting microsatellites in DNA. South African Institute of Computer Scientists and Information Technologists (SAICSIT) (2006), 111–120. [cited at p. 16, 17, 19, 63, 92]

[18] De Ridder, C., Reyneke, P., Watson, B. W., Reva, O., and Kourie, D. G. Cascading finite automata for minisatellite detection. In The 22nd Annual Symposium of the Pattern Recognition Association of South Africa (PRASA) (2011), pp. 31–36. [cited at p. 63, 92]

[19] Dix, A., Finlay, J., Abowd, G., and Beale, R. Human computer interaction. Pearson, 2004. [cited at p. 63, 94, 103, 104, 107, 108]

[20] Eason, K. Towards the experimental study of usability. In Behaviour and information technology (1984), vol. 3, pp. 133–143. [cited at p. 41, 44]

[21] Feller, J., and Fitzgerald, B. A framework analysis of the open source software development paradigm. In Proceedings of the twenty first international conference on Information systems (2000), Association for Information Systems, pp. 58–69. [cited at p. 9, 10, 49, 50]

[22] Fitzgerald, B. The transformation of open source software. Mis Quarterly (2006), 587–598. [cited at p. 48]

[23] Folmer, E., and Bosch, J. Architecting for usability: a survey. The Journal of Systems and Software 70 (2004), 61–78. [cited at p. 45, 46]

[24] Fonzo, V. D., Aluffi-Pentini, F., and Parisi, V. JSTRING: A novel java tandem repeats searcher in genomic sequences with an interactive graphic output. The Open Applied Informatics Journal, 2 (2008), 14–17. [cited at p. 20]

[25] Gusfield, D. Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, New York, NY, USA, 1997. [cited at p. 34, 35]

[26] Hars, A., and Ou, S. Working for free? motivations of participating in open source projects. In System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on (2001), IEEE, pp. 9–pp. [cited at p. 48]

[27] Henke, L., Fimmers, R., Josephi, E., Cleef, S., Dülmer, M., and Henke, J. Usefulness of conventional blood groups, DNA-minisatellites, and short tandem repeat polymorphisms in paternity testing: a comparison. Forensic Science International 103, 2 (1999), 133–142. [cited at p. 6]

[28] Hofstee, E. Constructing a good dissertation: A practical guide to finishing a Masters, MBA or PhD on Schedule. Sandton: EPE, 2006. [cited at p. 57, 60, 61, 65]

[29] Hornbak, K. Current practice in measuring usability: Challenges to usability studies and research. International Journal of Human-Computer Studies 64, 2 (Feb. 2006), 79–102. [cited at p. 41]

[30] Ilie, L. Factor oracle, suffix oracle. `www.csd.uwo.ca/faculty/ilie/Factor%20Oracle.ppt`. accessed: 21 November 2012. [cited at p. 34]

[31] ISO 9126. Software engineering—product quality–part 1: Quality model. International Organisation for Standardisation, 2001. [cited at p. 41]

[32] ISO 9241-11. Ergonomics requirements for office work with visual display terminals (vdts)–part 11: Guidance on usability. International Organisation for Standardisation, 1998. [cited at p. 41]

[33] Jacewicz, R., Berent, J., Prosniak, A., Dobosz, T., Kowalczyk, E., and Szram, S. Paternity determination of the deceased defendant in STR against RFLP analysis. International Congress Series (2004), 523 – 525. Progress in Forensic Genetics 10. [cited at p. 6]

[34] Jorda, J., and Kajava, A. V. T-REKS: identification of tandem REpeats in sequences with a k-means based algorithm. Bioinformatics 25, 20 (2009), 2632–2638. [cited at p. 20]

[35] Karaca, M., Bilgen, M., Onus, A. N., Ince, A. G., and Elmasulu, S. Y. Exact tandem repeats analyzer (E-TRA): A new program for dna sequence mining. Journal of Genetics 84, 1 (2005), 49–54. [cited at p. 20]

[36] Klencke, M., and Amsterdam, V. U. Advancements in Open Source Software Usability, 2005. [cited at p. 49, 51]

[37] Kolpakov, R., Bana, G., and Kucherov, G. mreps: Efficient and flexible detection of tandem repeats in DNA. Nucleic Acids Research 31, 13 (2003), 3672–3678. [cited at p. 18, 19, 23, 24, 25, 26, 27, 70]

[38] Kolpakov, R., and Kucherov, G. Finding maximal repetitions in a word in linear time. In Foundations of Computer Science, 1999. 40th Annual Symposium on (1999), pp. 596 –604. [cited at p. 35]

[39] Kolpakov, R., and Kucherov, G. Finding approximate repetitions under hamming distance. Theoretical Computer Science 303, 1 (2003), 135–156. [cited at p. 35]

[40] Kurtz, S., Choudhuri, J., Ohlebusch, E., Schleiermacher, C., Stoye, J., and Giegerich, R. REPuter: the manifold applications of repeat analysis on a genomic scale. Nucleic Acids Research 29, 22 (2001), 4633–4642. [cited at p. 35, 36]

[41] Kurtz, S., Ohlebusch, E., Schleiermacher, C., Stoye, J., and Giegerich, R. Computation and visualization of degenerate repeats in complete genomes. In ISMB (2000), Citeseer, pp. 228–238. [cited at p. 35, 36]

[42] Kurtz, S., and Schleiermacher, C. REPuter: fast computation of maximal repeats in complete genomes. Bioinformatics 15, 5 (1999), 426–427. [cited at p. 18, 20, 33, 35]

[43] Leclercq, S., Rivals, E., and Jarne, P. Detecting microsatellites within genomes: significant variation among algorithms. Bioinformatics 8, 125 (2007). [cited at p. 22, 57]

[44] Lefebvre, A., Lecroq, T., Dauchel, H., and Alexandre, J. FOR-Repeats: detects repeats on entire chromosomes and between genomes. Bioinformatics 19, 3 (2003), 319–326. [cited at p. 20, 36, 37]

[45] Leventhal, L., and Barnes, J. Usability engineering: process, products and examples. Pearson Prentice Hall, 2008. [cited at p. 42, 43, 47]

[46] Lewis, C., and Rieman, J. Task-Centered User Interface Design: A practical introduction. 1993. [cited at p. 94, 103]

[47] Ma, B., Tromp, J., and Li, M. PatternHunter: Faster and more sensitive homology search. Bioinformatics, 18 (2002), 440–445. [cited at p. 18]

[48] Mayer, C. Phobos—a tandem repeat search tool for complete genomes. `http://www.ruhr-uni-bochum.de/spezzoo/cm/cm_phobos.htm`, 10 2007. Accessed: Feb 2008. [cited at p. 19, 23]

[49] Mayer, C. Phobos ver 3.3.2 user manual: A tandem repeat search program, 2007. [cited at p. 23, 71, 79, 83]

[50] Mittal, P., and Singh, J. Merits and demerits of open source software. International Journal of Research in Computer and Communication Technology, 3 (March 2013). [cited at p. 8]

[51] Mokwana, D. Investigating software detecting minisatellites. Tech. rep., UNISA, 2013. [cited at p. 21]

[52] Mouton, J. How to succeed in your Master's and Doctoral studies. Van Schaik, 2001. [cited at p. 57]

[53] Mtsweni, J., and Biermann, E. Challenges & factors influencing the oss adoption rate in the SA government. In the proceedings of the track of the 2008 Free and Open Source Software for Geospatial Conference (FOSS4G2008). [cited at p. 9]

[54] Nichols, D., and Twidale, M. The Usability of Open Source Software. First Monday 8 (2003), 1–6. [cited at p. 49]

[55] Nichols, D., and Twidale, M. B. Usability and Open Source Software. Online `http://www.cs.waikato.ac.nz/~{}daven/docs/oss-wp.html`, 2002. accessed 25 May 2010. [cited at p. 9, 10, 49, 50, 51, 115]

[56] NIELSEN, J. Usability Engineering. Boston: Academic Press, 1993. [cited at p. 41, 42, 63, 94, 113, 127, 144, 146]

[57] NIELSEN, J. Heuristic Evaluations. In: J. Nielsen & R.L. Mack. (Eds), Usability Inspection Methods. New York: John Wiley & Sons, 1994. [cited at p. 63, 64]

[58] NORMAN, D. The design for everyday things. New York : Doubleday, 1990. [cited at p. 114]

[59] OLIVEROS, J. VENNY. an interactive tool for comparing lists with venn diagrams. Online: http://bioinfogp.cnb.csic.es/tools/venny/index.html, 2007. accessed 20 November 2012. [cited at p. 78]

[60] OLIVIER, M. Information Technology Research: A practical guide for Computer Science and Informatics, 2nd edition ed. Pretoria: Van Schaik, 2004. [cited at p. 57, 60, 61, 64]

[61] OPENUSABILITY. Online: http://www.openusability.org/. [accessed 11 October 2012]. [cited at p. 52]

[62] ÖZEL, B., GENÇER, AND STEPHENSON, C. An msc programme in open source information systems. In Towards Open Source Software Adoption (2006), OSS 2006 tOSSad workshop proceedings. [cited at p. 52]

[63] PANDE, S., AND GOMES, N. Article: Human resource information systems: A review in the adoption of open source. International Journal of Computer Applications 61, 8 (January 2013), 11–18. Published by Foundation of Computer Science, New York, USA. [cited at p. 8]

[64] RAYMOND, E. The cathedral and the bazaar: Musings on Linux and open source by an accidental revolutionary. O'Reilly Media, Inc., 1999. [cited at p. 48]

[65] RAZA, A. A Usability Maturity Model for Open Source Software. PhD thesis, The University of Wesstern Ontario, 2011. [cited at p. 44]

[66] RAZA, A., AND CAPRETZ, L. Contributor's preference in open source software usability: an emperical study. International Journal of Software Engineering and Applications 1, 2 (2010), 45–64. [cited at p. 49, 50]

[67] RAZA, A., CAPRETZ, L. F., AND AHMED, F. Improvement of open source software usability: an empirical evaluation from developers' perspective. Advances in Software Engineering 2010 (2010). [cited at p. 92]

[68] Reitmayr, E., Balazs, B., and Mühlig, J. Integrating usability with open source software development: Case studies from the initiative open usability. In Towards Open Source Software Adoption (2006), OSS 2006 tOSSad workshop proceedings. [cited at p. 52]

[69] Richardson, J., Ormerod, T., and Shepherd, A. The role of task analysis in capturing requirements for interface design. Interacting with Computers 9 (1998), 367–384. [cited at p. 103]

[70] Saha, S., Bridges, S., Magbanua, Z., and Peterson, D. Computational approaches and tools used in identification of dispersed recitative DNA sequences. Tropical Plant Biology, 1 (2008), 86–96. [cited at p. 16, 17, 18, 19]

[71] Schach, S. Object-Oriented and Classical Software Engineering. McGraw-Hill, 2007. [cited at p. 48]

[72] Shackel, B. Ergonomics in design for usability. In Human-Computer Interaction (1986). [cited at p. 41]

[73] Sharma, D., Issac, B., and Raghava, G. Spectral repeat finder (SRF): identification of repetitive sequences using fourier transformation. Bioinformatics, 20 (2004), 1405–1412. [cited at p. 18]

[74] Sharma, D., Issac, B., Raghava, G. P. S., and Ramaswamy, R. Spectral repeat finder (srf): identification of repetitive sequences using fourier transformation. Bioinformatics 20, 9 (2004), 1405–1412. [cited at p. 20]

[75] Sivess, V. Non-functional requirements in the software development process. Software Quality Journal 5, 4 (Dec. 1996), 285–294. [cited at p. 40]

[76] Smit, A., Hubley, R., and Green, P. Repeatmasker documentation. Online: `http://www.repeatmasker.org/webrepeatmaskerhelp.html`. accessed 25 May 2010. [cited at p. 17]

[77] Sonnhammer, E., and Durbin, R. A dot-matrix program with dynamic threshold control suited for genomic dna and protein sequence analysis. Gene, 167 (1995), 1–10. [cited at p. 18]

[78] South African Government. Criminal law (forensic procedures) amendment bill. Tech. rep., Minister for justice and constitutional development, December 2008. Government Gazette No. 31759. [cited at p. 9]

[79] Ssemugabi, S. Usability evaluation of a web-based e-learning application: a study of two evaluation methods. Master's thesis, UNISA, November 2006. [cited at p. 63, 118]

[80] Tawileh, A. Experiences in building a free and open source software training and certification programme. In Towards Open Source Software Adoption (2006), OSS 2006 tOSSad workshop proceedings. [cited at p. 52]

[81] Terry, M., Kay, M., and Lafreniere, B. Perceptions and practises of usability in the free/open source software (FoSS) community. ACM Press, New York, New York, USA, 2010, p. 999. [cited at p. 50, 51]

[82] The QCBS Wiki. Mining genomic data for tandem repeats. Online: http://qcbs.ca/wiki/bioinformatic_tools_to_detect_microsatellites_loci_from_genomic_data. accessed 10 May 2012. [cited at p. 19]

[83] Trudelle, P. Shall we dance? ten lessons learned from netscape's flirtation with open source ui development. Presented at the Open Source Meets Usability Workshop, Conference on Human factors in Computer Systems (CHI 2002), 2002. [cited at p. 50]

[84] Viorres, N., Xenofon, P., Stavrakis, M., Vlachogiannis, E., Koutsabasis, P., and Darzentas, J. Major HCI challenges for open source software adoption and development. Online Communities and Social Computing, 2nd International Conference, HCII 2007, pp. 455–464. [cited at p. 48, 49]

[85] Virtaneva, K., DÁmato, E., Miao, J., Koskiniemi, M., Norio, R., Avanzini, G., Franceschetti, S., Michelucci, R., Tassinari, C., Omer, S., Pennacchio, L., Myers, R., Dieguez-Lucena, J., Krahe, R., Chapelle, A., and Lehesjoki, A. Unstable minisatellite expansion causing recessively inherited myoclonus epilepsy, EPM1. Nature Genetics Journal 15, 4 (1997), 393–399. [cited at p. 6]

[86] Wexler, Y., Yakhini, Z., Kashi, Y., and Geiger, D. ATRHunter: Experimental results. Online: http://bioinfo.cs.technion.ac.il/atrhunter/ATRexperiments.htm. accessed 25 May 2010. [cited at p. 73]

[87] Wexler, Y., Yakhini, Z., Kashi, Y., and Geiger, D. Finding approximate tandem repeats in genomic sequences. In: Proceedings of the Eighth Annual International Conference on Computational Molecular Biology (2004), 223–232. [cited at p. 6, 18, 20, 30, 31, 32]

[88] Wexler, Y., Yakhini, Z., Kashi, Y., and Geiger, D. Finding approximate tandem repeats in genomic sequences. Journal of Computational Biology 12, 7 (2005), 928–924. [cited at p. 84, 89]

[89] Wirawan, A., Kwoh, C., Hsu, L., and Koh, T. INVERTER: Integrated variable numbER tandem repeat finder. In Computational Systems-Biology and Bioinformatics, vol. 115 of Communications in Computer and Information Science. Springer Berlin Heidelberg, 2010, pp. 151–164. [cited at p. 20]

[90] Ye, Y., and Kishida, K. Toward an understanding of the motivation of open source software developers. International conference on software engineering. [cited at p. 9, 10, 49, 50]

[91] Zhao, L., and Deek, F. Exploratory inspection - a learning model for improving open source software usability. In: CHI '06 extended abstracts on Human factors in computing systems. CHI EA '06. [cited at p. 51]

[92] Zhou, Y., and Mishra, B. Quantifying the mechanisms for segmental duplications in mammalian genomes by statistical analysis and modeling. In: Proceedings of the National Academy of Sciences of the United States of America 102, 11 (March 2005), 4051–4056. [cited at p. 9, 10, 50]

Appendices

Appendix A

Usability Test Plan

A.1 Document Overview

This document describes a test plan for conducting a usability test of FireμSat Graphic User Interface (GUI) and FireμSatPlus prototype. The goals of this usability testing include identifying potential design concerns to be addressed in order to improve the efficiency, productivity, and end-user satisfaction.

The usability test objectives are:

- To determine design inconsistencies and usability problem areas within the user interface and content areas. Potential sources of error may include:
  - Navigation errors – failure to locate functions, excessive keystrokes to complete a function, failure to follow recommended screen flow.
  - Presentation errors – failure to locate and properly act upon desired information in screens, selection errors due to labelling ambiguities. Control usage problems – improper toolbar or entry field usage.
- Use the user interfaces under controlled test conditions with representative users. Collected data will be used to access whether usability goals regarding an effective, efficient, and well-received user interface have been achieved.

- Establish baseline user-satisfaction levels of the user interface for future usability evaluations.

FireμSat and FireμSatPlus are developed for computational biologist who are not computer experts. In this test, computer science students will be used to mimic computational biologists. The test will be conducted with about 5 to 8 participants.

A.2 Purpose

This test will evaluate FireμSat and FireμSatPlus user interfaces based on Nielsen's Ten heuristics [56] as summarised in Chapter 6.

A.3 Methodology

This test will involve two separate groups of five to eight participants. One group will test the FireμSat and the other will test the FireμSatPlus prototype. Each participant will be ask to complete predefined tasks on the UI.

At the end of the test, each participant is requested to complete a questionnaire about his experience on using the GUI. Finally the questionnaire will be analysed and the results be reported.

A.3.1 Participants

The participants will be recruited orally based on the following questions:

- Do you or have you studied computer science?
- Do you have ability to use a computer?
- Are you interested in participation in this usability study?
- Are you available to participate in usability study?

The participants' responsibilities will be to attempt to complete a set of representative task scenarios presented to them in as efficient and to provide feedback regarding the usability and acceptability of the user interface. The participants will be directed to provide honest opinions regarding the usability of the UIs, and to participate in post-session subjective questionnaires.

No experience on performing these tasks as training will be provided.

A.3.2 Training

Participants will be given a brief training on tandem repeats and their detection background. The participants will receive an overview of the usability test procedure, equipment—input data—and software.

A.3.3 Procedure

Participants will be given a copy of the relevant GUI for testing. The testing will happen on participants' own computers on their own time, so as to follow real Academic/Research Software (ARS) and/or Open Source Software (OSS) development and testing practises. Within OSS development environment, software is released to the user community who use it and report bugs to developers as part of testing.

The facilitator will brief the participants on the application and instruct the participant that they are evaluating the application, rather than the facilitator evaluating the participant. Participants will sign an informed consent that acknowledges that: the participation is voluntary, that participation can cease at any time, and their privacy of identification will be safeguarded. A sample consent form appears in Appendix B. The facilitator will ask the participant if they have any questions and how to be contacted in case that participants requires additional help.

After completing the tasks, the participant will complete the post-task questionnaire.

A.4 Usability tasks

Refer to Appendix C for tasks to be completed.

A.5 Usability metrics

Usability metrics refers to user performance measured against specific performance goals necessary to satisfy usability requirements. Subjective evaluations against Nielsen's guidelines will be used in this test.

A.5.1 Subjective evaluations

Subjective evaluations regarding user's perception of the UI will be collected via questionnaire. The questionnaire is based on ten guidelines identified by Nielsen [56].

A.6 Usability goals

The next section describes the usability goals for this study.

A.6.1 Subjective measures

Subjective opinions about specific tasks, features, and functionality will be surveyed. The results of each group will be analysed and areas that do not meet the Nielsen's guidelines be identified. At the same time, the results of the two groups will be analysed to identify which of the two software application comply more to the guidelines.

A.7 Reporting results

The Usability Test Report will be provided at the conclusion of the usability test. It will consist of a report and/or a presentation of the results; evaluate the usability metrics against the pre-approved goals, subjective evaluations, and specific usability problems.

Appendix B

Consent Form

Evaluation of the FireμSat and FireμSatPlus
at
University of South Africa

Student consent form

I ____ (First name and Surname) and student number _______________, a student at University of South Africa state that

- I have not been put under any pressure to participate in this evaluation exercise, and
- have willingly participated in it.
- I know that my participation can cease at any time I want.
- I realise that the findings of the evaluation will be used for research purposes, and
- that the findings will be published without revealing my identity.

Signed _______________________ date ________________________

Appendix C

Tasks

The following tasks have to be completed by following appropriate instructions.
`Assumptions:` The input sources file and the output file have been selected before commencement of each task.

Task 1: `Purpose:` To investigate the easy to use the Graphic User Interface (GUI) design.

- Use the GUI default values to search for all Perfect minisatellites.

Task 2: `Purpose:` To investigate the advance use of the system GUI.

- Search for minitsaltellites with the following parameters:

  ```
  Min. motif size: 7
  Max. motif size: 12
  TR type: Approximate
  Deletion penalty: 0.5
  Insertion penalty: 0.5
  Mismatch penalty: 0.1
  ```

Task 3: `Purpose:` To investigate user control of the system GUI.

- Use parameters as in task 2 and start the search search. Stop the search before it completes and include the `Perfect TRs` in the search.

Task 4: `Purpose:` To investigate error handling.

- Use parameters as in task 3 but change the `Max motif` to `5`. Start the search.

Appendix D

Questionnaire

Name: ____

Visibility of System Status
The system keeps me informed through feedback about what it is doing.
Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐
For every action I make, I can see or hear the results of that action.
Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐
Comment:

Match between System and the Real World
The language used is natural, since the terms, phrases and concepts are similar to those used in my day-to-day working environment.
Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐
I am not confused by the use of terms.
Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐
Comment:

User control and freedom

I control the system, rather than it controlling me.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐

The system works the way I want it to work.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐

Comment:

Consistency and adherence to Standards

The same convention (words) is used throughout the GUI.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐

The convention used on the GUI is similar to the ones on other GUI I am use to.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐

Comment:

Error Prevention

The system supports me in such a way that it is not easy to make serous mistakes.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐

Whenever a mistake is made an error message is given.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐

Comment:

Recognition rather than Recall

There is an obvious relationship between controls and their actions.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐

Options to be selected are easy to recognise.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ Strongly disagree ☐

Comment:

Flexibility and efficiency of use
The GUI caters for different levels of users, from novice to expert.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐

There is an option to use the keyboard alone to perform tasks.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐

Comment:

Aesthetic and Minimalistic Design
The information on GUI is not too much to confuse or distract me.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐

There are no excessive use of graphics and images.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐

Comment:

Help Users Recognise, Diagnose and Recover from Errors
Error messages are expressed in plain language.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐

Each error message gives information on how to fix the error.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐

Comment:

Help and Documentation
I find the online help facilities âĂŞ useful.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐

Links to other resources are helpful.

Strongly agree ☐ *Agree* ☐ *Maybe* ☐ *Disagree* ☐ *Strongly disagree* ☐

Comment:

Appendix E

FireμSatPlus GUI source code

main.cpp file:

```
#include <QtGui/QApplication>
#include "FireMuSatPlus.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    FireMuSatPlus w;
    w.show();

    return a.exec();
}
```

FireMuSatPlus.h file:

```
#ifndef FIREMUSATPLUS_H
#define FIREMUSATPLUS_H

#include <QMainWindow>
#include <QtGui/QFileDialog>
#include <QFutureWatcher>

namespace Ui {
    class FireMuSatPlus;
}

class FireMuSatPlus : public QMainWindow
{
    Q_OBJECT

public:
    explicit FireMuSatPlus(QWidget *parent = 0);
    ~FireMuSatPlus();

public slots:
   // int busy();
    void searchStatus();
    void adjWinSize();
    bool parOk();

private slots:
    void on_toolButton_clicked();
    void on_toolButton_2_clicked();

    void on_btnHelp_clicked();

private:
    Ui::FireMuSatPlus *ui;
    QFutureWatcher<void> FutureWatcher;

};

#endif // FIREMUSATPLUS_H
```

FireMuSatPlus.cpp file:

```
#include "FireMuSatPlus.h"
#include "ui_FireMuSatPlus.h"
#include <QtGui/QMessageBox>
#include <QtGui/QProgressDialog>
#include <QtGui/QProgressBar>
#include <QtGui/QFileDialog>

FireMuSatPlus::FireMuSatPlus(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::FireMuSatPlus)
{
    ui->setupUi(this);
    ui->extension->hide ();
    this->adjustSize ();

}

FireMuSatPlus::~FireMuSatPlus()
{
    delete ui;
}

void FireMuSatPlus::searchStatus (){
    if(parOk()){
    int numTRs = 99990;
    QProgressDialog progress("Searching...", "Stop search", 0,numTRs, this);
         progress.setWindowModality(Qt::WindowModal);
           // progress.setRange(0,0);
         progress.show();
         for (int i = 0; i <= numTRs; i++) {
             progress.setValue(i);

             if (progress.wasCanceled()){
                 QMessageBox::StandardButton ret;
                 ret = QMessageBox::warning(this, tr("Confirm cancel"),
                                                  tr("The search process has been
stopped.\n" "Press Yes to cancel the search?"),

QMessageBox::No | QMessageBox::Yes);
                 if (ret == QMessageBox::Yes)
```

```
                            break;
                    else if (ret == QMessageBox::No){
                            progress.reset();
                            progress.show();
                    }

                }
                //... search
               qApp->processEvents();
            }
            if(!progress.wasCanceled()){
            QMessageBox::StandardButton ret2;
            ret2 = QMessageBox::information(this, tr("FireMuSatPlus"),
                                       tr("Searching complete.\n"
                                       "Press ok to view results?"),
                                        QMessageBox::Ok);
            }
        }
}




bool FireMuSatPlus::parOk()
{
    if(!ui->grbRange->isChecked())
        return TRUE;
    else if(ui->spbMin->value() >ui->spbMax->value()){
       QMessageBox::warning(this, tr("Invalid input"), tr("Minimum motif
length must be
less than maximum motif length"), QMessageBox::Ok);
        return FALSE;}
    else
        return TRUE;
}

void FireMuSatPlus::adjWinSize ()
{
    if (!ui->btnMore->isChecked ()){
        this->resize (550, 16777215);
        this->adjustSize ();
    }
}


void FireMuSatPlus::on_toolButton_clicked()
{
    QString inputDialog;
    inputDialog = QFileDialog::getOpenFileName (this, tr("Open File"),
  QDir::currentPath (), tr("Files(*.txt ;;All Files (*)"));
  ui->ldtInput->setText (inputDialog);

}





void FireMuSatPlus::on_toolButton_2_clicked()
{
    QString saveDialog = QFileDialog::getSaveFileName (this, tr("Save as"),
                           QDir::currentPath (), tr("Files (*.txt)"));
  yui->ldtOutput->setText (saveDialog);
}

void FireMuSatPlus::on_btnHelp_clicked()
{
    QMessageBox::about(this,"FiremuSatPlus",
                    "This app for detecting microsaltellites and minisatellites
in a DNA sequence."
"\nMore inforation on parameters are
available at www.dna-algo.co.za\n"
                                        "Bye.\n");
```

List of Symbols and Abbreviations

Abbreviation	Description	Definition
DNA	DeoxyriboNucleic Acid	page 5
TR	Tandem Repeat	page 5
ATR	Approximate Tandem Repeat	page 6
PTR	Perfect Tandem Repeat	page 6
ARS	Academic/Research Software	page 10
OSS	Open source software	page 10
PS	Proprietary software	page 9
GUI	Graphic User Interface	page 12
ISO	International Organisation for Standardisation	page 44
HCI	Human Computer Interaction	page 47

List of Figures

List of Tables