FireμSat: An Algorithm to Detect Microsatellites in DNA

Corné de Ridder[1], Derrick G. Kourie[2], and Bruce W. Watson[2]

[1] School of Computing, University of South Africa, South Africa, Pretoria 0003
[2] Fastar Research Group, Department of Computer Science, University of Pretoria, South Africa, Pretoria 0002
driddc@unisa.ac.za, dkourie@cs.up.ac.za, watson@cs.up.ac.za

Abstract. In the context of this paper microsatellites (short *approximate* tandem repeats) refer to consecutive patterns contained in genomic sequences. A new algorithm to detect such microsatellites in DNA is proposed. The algorithm relies on the construction of finite automata originating from the Moore machine paradigm. The proposed finite automata contain "counting states". The overall algorithm is designed to support user requirements as expressed by the typical geneticist.

Keywords: finite automata, microsatellites, tandem repeats, computational biology

1 Introduction

A perfect tandem repeat (PTR) is a string of nucleotides in a genomic sequence whose initial substring (of some arbitrary length) is followed by two or more copies of that substring. The introductory substring is called the motif of the PTR.

For example, `ACGACGACGACGACG` is a PTR, in which the motif is a substring of length 3 (*i.e.* $|motif| = 3$), namely `ACG`.

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

An example of an ATR is: `ACGACTACG`. In this case, the substring `ACT` that directly follows on the motif `ACG` has a mismatch in the third position.

In the absence of further qualification, reference to a TR should be construed as a reference to either a PTR or an ATR. It should be noted that there is not complete consensus on the precise meaning of a TR. In some cases, it is not required that the TR starts off with its motif. In fact, there are some who would be content to regard a string of approximate repeats as a TR, even if it did not contain the motif at all. However, this research has been driven by the requirements of geneticists and molecular biologists who were interested in detecting TR's as defined above.

A TR element (TRE) that matches the identified motif of the TR will be referred to as a PTR element (PTRE). A TRE that does not match the motif is referred to as an ATR element (ATRE).

In the literature ([21]; [15]; [16];) a distinction is made between TR's that constitute microsatellites, minisatellites and satellites. However, terminology is not used consistently in the literature.

Castelo et al. [4] coins the term *Simple Sequence Repeats (SSR's)* for microsatellites; Tran et al. [22] terms microsatellites *short tandem repeats*. Delgrange and Rivals

[6], Benson [3] and Abajian [1] consider TR's characterized by motif lengths greater than or equal to two and smaller than or equal to five ($2 \leq |motif| \leq 5$) to be microsatellites. Thurston and Field [21] consider TR's microsatellites if $2 \leq |motif| \leq 6$. For the purposes of this paper microsatellites will be consider to be TR's with a PTRE or motif such that $2 \leq |motif| \leq 5$. Microsatellites may include both PTRE *and* ATRE.

Although the algorithm, FireμSat, that is proposed here, can in theory be applied to search for TR's of any length, the focus at this stage, is to introduce an algorithm that searches specifically for microsatellites.

It will be seen that FireμSat has several parameters that can be used to tune its search. It should be emphasised that these parameters have been devised in consultation with the intended user community, who have been unable to usefully deploy existing software for TR detection. The objective is to fine tune a TR search so that redundant data is avoided, and relevant data is not missed.

The remainder of this paper is laid out as follows. Section 2 provides a brief overview of existing software packages or proposed algorithms that attempt to address the computational problem of detecting microsatellites on DNA. Section 3 defines the problem in a formal manner. In section 4 an outline is provided of how finite automaton (FA) technology can be used to detect tandem repeats in a DNA string, culminating in pseudo-code for the FireμSat algorithm. Section 5 concludes the paper, and points to work currently underway to empirically test the FireμSat algorithm.

2 Related work

There are various software packages that search directly or indirectly for microsatellites. In this regard Van den Bergh [23] mentions that although most authors reference a selection of software that has been developed before the software that they propose, there does not seem to be a comprehensive catalogue of relevant software. It is possible to classify existing software in various ways. Benson [3] divided the algorithms that he investigated into three categories and mentions their shortcomings as follows:

- *Alignment algorithms*
  Alignment algorithms proposed by (Benson [2], Kannan and Myers [7] and Schmidt [19]) have an excessive running time—their running time is exponential.
- *Algorithms from the field of data compression*
  An algorithm proposed by Milosavljevic and Jurka [14] detects simple sequences thus mixtures of fragments that occur elsewhere. Simple sequences may or may not contain TR's. This algorithm makes no attempt to deduce a repeated pattern. Rivals et al. [17] also developed an algorithm belonging to this category that is based on the presence of preselected patterns with ($1 \leq |motif| \leq 3$). This algorithm suffers from severe limitations in terms of the motif length that is allowed, and in terms of the fact that the algorithm only searches for preselected motifs.
- *Algorithms that aim to find TR's more directly.*
  Of these algorithms, the one developed by Landau et al. [11] is limited by its definition of approximate repeats. The algorithm requires that two copies differ by k or fewer substitutions (Hamming distance) or by k or fewer substitutions and indels (unit cost edit distance). The requirement for a fixed number of differences rather than a percentage is regarded as unsatisfactory. Similarly, the treatment of substitutions and indels as equals is regarded as unsatisfactory. The heuristic

algorithm proposed by Karlin et al. [8] is hampered in the same manner by the use of matching blocks separated by error blocks of fixed size. Myers and Sagot [15] has proposed an exact algorithm that requires that the approximate pattern size and a range for the number of copies should be pre-specified. An earlier algorithm of Benson [2] only finds TR's if they have a pattern size that is specified in advance.

Delgrange and Rivals [6] argue that an exact algorithm that entails the systematic detection of significant TR's in a way that is independent of the motif or of the sequence length is beyond the scope of present methods. In regard to existing software Delgrange and Rivals [6] also distinguish between three different classes of algorithms and their shortcomings as follows:

- *Fast algorithms from the field of computer science.*
  In the field of computer science there are several fast algorithms that search for only two exact tandem repeats. Authors presenting these approaches include Main and Lorentz [13]; Kolpakov and Kucherov [9] and Stoye and Gusfield [20]. Although these algorithms may be useful as filters to detect possible duplicate motifs they do not comply with the needs of molecular biologists [6].
- *Algorithms that do not make provision for the detection of TR's containing substitutions, deletions and insertions at once.*
  Algorithms in this category include those developed by Kolpakov and Kucherov [10], as well as the algorithms developed by Landau et al. [12] and Coward and Drablos [5]. These particular algorithms only make provision for substitutions.
- *Algorithms that detect TR's and allow for substitutions, insertions and deletions.*
  These algorithms include the work of Myers and Sagot [15] who introduced a combinatorially exhaustive approach that identifies several possible motifs and alignments for each TR. The complexity of this approach depends exponentially on some parameters. The work of Rivals [18] is limited to small motifs and allows only indels between two of the motifs within a TR.

3 Formal problem statement

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

However, when running FireμSat, the user is required to choose a maximum value for ε that complies with certain practical considerations. In determining whether the string $\rho u_2 \cdots u_p$ is to be construed as an ATR, this value of ε represents the maximum number of mutations (or errors) that are tolerated, in deciding whether or not, for each $k = 2 \cdots p$, u_k represents an acceptable ATRE. In 3.1, the author discusses the types of mutations that are tolerated. Here it is emphasized that the following toleration limits on ε apply for a given ρ.

1. If $|\rho| = 2$ or $|\rho| = 3$ then only zero or one error is tolerated; i.e. ε may be chosen as either 0 or 1. (The default is 1.)
2. If $|\rho| = 4$ or $|\rho| = 5$ then zero, one or two errors are allowed; i.e. ε may be chosen as either 0, 1 or 2. (The default is 2.)

Recall that the objective is to detect microsatellites. This means that $2 \leq |\rho| \leq 5$.

Consider an example where $\rho =$ ACGTT. Then $|\rho| = 5$ and the user may consequently select the maximum number of errors to be either 0 or 1 or 2. If the user selects "2", then ACT would be regarded as an ATRE, since it may be construed as the motif in which two deletions (see 3.1) have occurred. Likewise, ACGT could be regarded as an ATRE, since it may be seen as the motif in which one deletion has occurred. (See 3.1.) However, AC will not be regarded as an ATRE.

3.1 Type of mutations tolerated

A substring u is considered similar to the substring ρ^p if it can be written as $u = u_1 u_2 \cdots u_p$ where each word u_k $(k = 1 \cdots p)$ is obtained by at most ε mutations on ρ and where ε is some pre-specified limit in the range $0 \leq r \leq |\rho|$. This was explained in the previous paragraph (3). (Note that in running FireμSat, the user has further options for constraining the search for ATR's. These options are discussed in 3.2. They are concerned with constraining the ratio of ATRE's to PTRE's in a string and/or constraining the number of consecutive ATRE's in the string.)

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

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

It should be noted that all these words keep at least 2 bases from the original word ρ. As it stands, the foregoing could lead to ambiguity in determining the mutational origin of a string. For example, ACG could be construed as some intended PTRE, ρ, or as a deletion of the last nucleotide, G, of the PTRE ρ, followed by the insertion of G.

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

1. A string will be interpreted as a PTRE rather than as an ATRE with mutations.
2. A string will always be regarded as an ATRE that results from mismatches, rather than from insertions or deletions.
3. An ATRE will be regarded as originating from a deletion rather than from an insertion.

This manner of defining authorized forms of mismatches and deletions of u_k derives from experimental observations cited by Rivals et al. [17]. It has been endorsed by Benson

[3] as providing statistically relevant information. The algorithm proposed also allows for other types of errors that can be adjusted by the user. More details pertaining to this matter are to be found in 3.2.

Indeed, the approach was discussed with a molecular biologist, L. P. Wright, from the University of Pretoria, who was positive about the statistical relevance of the information that would be generated by the proposed algorithm.

In principle, then, an algorithm seeking TR's could rely on the motif error (ε) alone to determine when the end of a candidate string has been found. However, in practice, it is useful to rely on additional metrics. In 3.2 three such metrics are introduced. They determine whether a string that has been found to be a possible TR at some point in the algorithm, should be output as such, or whether further processing should occur to see if the string can be further extended to produce a longer TR.

3.2 Additional metrics and threshold values

In addition to considering ε (the maximum motif error that may occur within a TR), FireμSat also computes three additional metrics. These are σ, the so-called substring error; tn_atreC, the number of ATRE's that occur consecutively; and tn_tre, the total number of TRE's. In each case, the user can specify maximum values for these metrics, which FireμSat will use as a threshold value in determining when a given substring can be regarded as a TR. Each of these metrics will now be considered in turn.

1. *The substring error*:σ
   This is a measure of the extent to which the number (weighted as described below) of ATRE's in the candidate TR exceeds the number of PTRE's. The measure is computed at appropriate points by FireμSat and then compared against a user-specified threshold value of the *maximum substring error allowed*, τ. During processing $\sigma \leq \tau$ should always hold.
   In line with the guidelines suggested by Benson [3], the value of σ depends, *inter alia* on penalties (or weights) allocated by the user to mismatches (p_m), deletions (p_d) and insertions (p_i). For a given motif, ρ, and a given substring that has been partitioned into the form $u = \rho u_2 \cdots u_p$, σ on u is computed as:

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

   where n_d is the number of deletions in u; n_i is the number of insertions in u; n_m is the number of mismatches in u; and n_ptre is the number of PTRE's in u. The user may rely on system default values for the penalties. These are $p_i = 1.0$, $p_d = 1.0$ and $p_m = 0.5$ respectively. A penalty weight of 0 may be chosen for one or more of the mutation types, in which case no penalty is assigned to ATRE's that derive from that mutation type.
   The value of σ therefore reflects the extent to which the number of ATRE's exceeds the number of PTRE's, weighted in terms of penalty values associated with mismatches, deletions and insertions.
   The foregoing implies that FireμSat has to keep a count of the number of the various types of mutations. As will be seen in section 4.1, FireμSat makes use of an FA denoted by FA_{TR}, which is the sum (in the sense of Kleene's theorem Rule 2 of part 3) of four other FA's: one for recognizing PTRE's, and one each for recognizing insertions, deletions and mismatches. In general the substring error σ is calculated every time a final state is reached in FA_{TR}. Each such final state is associated with a unit increment in either the number of PTRE's, or the number of insertions, or the number of deletions or the number of mismatches. It is these final states, therefore, that enable the counting of the various types of mutations.

For as long as $\sigma \leq \tau$ holds, the scan of the input string continues in an effort to increase the length of the TR found to date. If the condition is not met, then the TR found to date is output, and the next TR in the input string is sought.
Of course, whenever a dead end state (a state that has only incoming edges, including a loop into the state itself labelled with all the alphabet letters of the input alphabet) of FA_{TR} is reached, then the TR is also output, and the search for the next TR resumed.

2. *The maximum number of consecutive ATRE's* (α)
The user has the option of entering a value denoted by α. This value indicates the maximum number of ATRE's that are allowed to occur next to each other. Thus α serves as a second threshold value.
If the user specifies a value for α, then the counter *tn_atreC* is maintained to record the total number of consecutive ATRE's since the last PTRE. The counter is incremented whenever an ATRE has been read (indicated by a transition to a final state of FA_{TR}) irrespective of the type of elements—whether it be an insertion, deletion or mismatch. However, when a PTRE is read, then the value of *tn_atreC* is again set to zero. The processing of a string will only proceed if $\alpha \leq$ tn_atreC.
Note that a value for α is not activated by default. Thus, if the user does not enter a value for α, then there is no limit to the number of ATRE's that may occur consecutively. (Alternatively, one might say that the default value of α is ∞.)
3. *The minimum number of tandem repeat elements (β)*
To avoid the output of unwanted data, the user may indicate the minimum number of TRE's that has to occur before a TR is output, denoted by β. To this end, a count, *tn_tre*, is kept of the total number of tandem repeat elements encountered to date in the current candidate TR. In fact, a count is also kept of the total number of PTRE's encountered to date, *tn_ptre*, and of the total number of ATRE's encountered to date, *tn_atre*. Clearly, $tn_tre = tn_ptre + tn_atre$.
The current candidate TR will only be reported as a TR if one of the previously mentioned thresholds or terminating conditions is encountered *and* if $tn_tre \geq \beta$. The default value for β is two.

To illustrate these concepts, consider the genetic substring `ACGACACACACGCGCGACGACT`. Let the motif be `ACG`. The values for *n_d,n_i, n_m, tn_ptre, tn_atre* and *tn_atreC* are as follows at different processing intervals of the substring.

0.	ACGACACACACGCGCACGACT	n_d	n_i	n_m	tn_ptre	tn_atre	tn_atreC
1.	ACG	0	0	0	1	0	0
2.	ACGAC	1	0	0	1	1	1
3.	ACGACAC	2	0	0	1	2	2
4.	ACGACACAC	3	0	0	1	3	3
5.	ACGACACACACG	3	0	0	2	3	0
6.	ACGACACACACGCGC	3	0	1	2	4	1
7.	ACGACACACACGCGCGACG	3	0	1	3	4	0
8.	ACGACACACACGCGCACGACT	3	0	2	3	5	3

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

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

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

The algorithm is invoked by:

$$\text{Fire}\mu\text{Sat}(min,\ max, \varepsilon, \tau, \alpha, \beta, p_d, p_i, p_m, gSeq)$$

where *gSeq* represents the entered genetic sequence. It returns all TR's with motif lengths in the range [*min,max*] in *gSeq*, subject to motif error ε and threshold values τ, α and β as discussed in subsections 3.1 and 3.2 respectively.

4 Algorithm Construction

4.1 FireμSat

The theory underlying FireμSat is a combination of straightforward FA technology combined with a flavour of Moore machine technology. How this theory is applied will be elaborated in the process of introducing the theoretical underpinnings of FireμSat.

For illustrative purposes, `ACG` will be used throughout as the motif string. In addition, to facilitate the explanation of the algorithm, the following FA's are introduced. In each case, the way in which the given FA scans a string of the form $u = \rho u_2 u_3 \cdots u_p$ will be described.

- $FA_P(\rho)$ is an FA that reaches a final state after scanning the first occurrence of ρ in u. In fact, it reaches the final state again if $u_2 = \rho$ is encountered in u, and again if $u_3 = \rho$ is encountered in u, etc. However, $FA_P(\rho)$ goes to a dead end state as soon as a character in u is encountered that indicates that u is not a PTR. Thus, $FA_P(\rho)$ accepts a PTR of arbitrary length, with motif ρ, entering the final states as many times as there are PTRE's in the PTR.
- $FA_D(\rho, \varepsilon)$ is an FA that, upon scanning u, reaches its first final state once the substring ρ has been read. $FA_D(\rho, \varepsilon)$ continues to reach final states after scanning each word, u_i (where $i = 2 \cdots p$) provided that one of the following conditions hold: a) either $u_i = \rho$ or b) u_i is a word deduced from ρ that contains a maximum of ε deletions.
- $FA_M(\rho, \varepsilon)$ is an FA that functions analogously to $FA_D(\rho, \varepsilon)$, except that it functions in terms of *mismatches* rather than deletions.
- $FA_I(\rho, \varepsilon)$ is an FA that functions analogously to $FA_D(\rho, \varepsilon)$, except that it functions in terms of *insertions* rather than deletions.
- $FA_{TR}(\rho, \varepsilon)$ is an FA obtained from the sum of all the previously defined FA's. Thus:
$$FA_{TR}(\rho, \varepsilon) = FA_P(\rho) + FA_D(\rho, \varepsilon) + FA_M(\rho, \varepsilon) + FA_I(\rho, \varepsilon)$$
- Finally, the predicate $isTR(gSeq[i, j], \varepsilon, \tau, \alpha, \beta, \rho)$ is defined as true if there is a TR with motif ρ in the genetic sequence $gSeq[i, j]$, such that the motif error is no greater than ε, the substring error (σ) is no greater than τ, the number of consecutive occurrences of ATRE's (tn_atreC) is no greater than α and the total number of TRE's (tn_tre) is at least β.

The *Fire Engine* software [24] constructs an FA from a regular expression (r.e.) that is provided as input. For a given motif, it is relatively easy to specify the regular expressions that correspond to the various FA's just mentioned above.

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

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

If these deterministic FA's ($FA_D(\texttt{ACG},\texttt{1})$, $FA_M(\texttt{ACG},\texttt{1})$) are constructed and distinction is made between the type of final states a trace through each respective FA will confirm that strings of the form $\rho u_1 \cdots u_q$ are recognized, where:

- $\rho u_1 \cdots u_q$ may be preceded by p, some arbitrary non-motif prefix.
- ρ is the motif (in the present example, ACG) of the TR,
- each $u_i, i = 1 \cdots q$ is an ATRE based on ρ, allowing for an error of maximally ε. Note that in the present example, $\rho = \texttt{ACG}$ and thus $|\rho| = 3$. Therefore, as previously discussed, ε is only allowed to assume the value of 1 or 0.
- $q \geq 1$ is the number of ATRE's that follow on from the motif in the TR.

If $FA_D(\texttt{ACG},\texttt{1})$, $FA_M(\texttt{ACG},\texttt{1})$ and $FA_I(\texttt{ACG},\texttt{1})$ are constructed as deterministic FA's then it will be seen that if any additional element that does not belong to the TR identified up to that point is encountered in the input string, the FA will transit to a dead-end state in each respective case.

It is possible to distinguish between two kinds of final states in each of these machines: those which signal that a motif (PTRE) has been scanned, and those which indicate that a deletion or mismatch or insertion has been scanned. These states will be referred to as PTR- and ATR-final states, respectively. As explained below, the number of transitions into these states have to be counted, and the respective values of σ, tn_atreC and tn_tre have to be correspondingly updated so as to ensure that the strings designated as TR's are consistent with thresholds τ, α and β respectively, as explained in section 3.2 above.

In order to contribute to the foregoing explanation figure 1 has been included. A trace through figure 1 will confirm that strings of the form $p\rho u_1 \cdots u_q$ are recognized, where:

- p is some arbitrary non-motif prefix preceding a TR,
- ρ is the motif (in the present example, ACG) of the TR,
- each $u_i, i = 1 \cdots q$ is a TRE based on ρ, allowing for an error of maximally ε. Note that in the present example, $\rho = \texttt{ACG}$ and thus $|\rho| = 3$. Therefore, as previously discussed, ε is only allowed to assume the value of 1. Thus only 1 deletion may occur.
- $q \geq 1$ is the number of ATRE's that follow on from the motif in the TR.

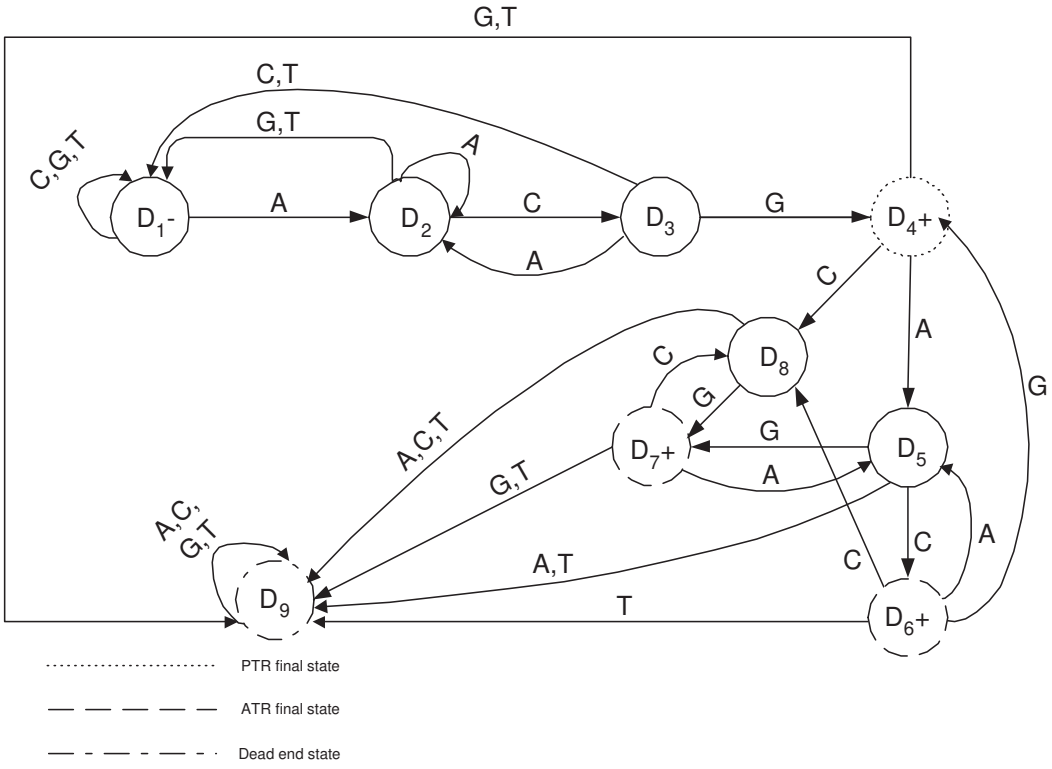

Figure 1. $FA_D(\texttt{ACG},1)$

It will also be seen that if any additional element that does not belong to the TR identified up to that point is encountered in the input string, then the FA transits to a dead-end state. State D_9 in figure 1 is a dead-end state without any outgoing edges.

In figure 1, there are also two kinds of final states: those which signal that a motif (PTRE) has been scanned, and those which indicate that a deletion has been scanned. These states will be referred to as PTR- and ATR-final states, respectively. As explained below, the number of transitions into these states have to be counted, and the respective values of σ, tn_atreC and tn_tre have to be correspondingly updated so as to ensure that the strings designated as TR's are consistent with thresholds τ, α and β respectively, as explained in section 3.2 above.

In order to construct $FA_{TR}(\rho, 1)$ we first construct the respective constituent machines and then apply the constructive algorithm which forms part of the proof of *Rule 2, Part 3* of Kleene's theorem.
$FA_{TR}(\texttt{ACG}, 1) = FA_P(\texttt{ACG}) + FA_D(\texttt{ACG}, 1) + FA_M(\texttt{ACG}, 1) + FA_I(\texttt{ACG}, 1)$.
This can be done by using the *Fire Engine* software toolkit [24] that provides a function for adding FA's.

The discussion to date can be generalized: $FA_{TR}(\texttt{XYZ}, 1)$ is an FA for recognizing the parameterized motif XYZ of length 3. The parameters are X, Y and Z and each of these parameters can be instantiated to any one of the nucleotides $\{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$. This parameterized FA is, as before, the sum of four other FA's, each of which are also parameterized.

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

- $FA_D(\texttt{XYZ}, \texttt{1})$ accepts the language defined by the r.e. $(\texttt{XYZ})(\texttt{XYZ} + \texttt{XY} + \texttt{XZ} + \texttt{YZ})^*$.
- $FA_M(\texttt{XYZ}, \texttt{1})$ accepts the language defined by the r.e. $(\texttt{XYZ})(\texttt{XYZ} + \texttt{YYZ} + \texttt{ZYZ} + \texttt{RYZ} + \texttt{XXZ} + \texttt{XZZ} + \texttt{XRZ} + \texttt{XYX} + \texttt{XYY} + \texttt{XYR})^*$.
- $FA_I(\texttt{XYZ}, \texttt{1})$ accepts the language defined by the r.e. $(\texttt{XYZ})(\texttt{XYZ} + \texttt{XXYZ} + \texttt{YXYZ} + \texttt{ZXYZ} + \texttt{RXYZ} + \texttt{XYYZ} + \texttt{XZYZ} + \texttt{XRYZ} + \texttt{XYXZ} + \texttt{XYZZ} + \texttt{XYRZ} + \texttt{XYZX} + \texttt{XYZY} + \texttt{XYZR})^*$.

Thus, in principle, any FA_{TR} of motif length 3 can be algorithmically constructed. Similarly, parameterized versions for $FA_{TR}(\rho, \varepsilon)$ can be constructed for $|\rho| = 2, 4, 5$ and for permissable values of ε. In each case, the r.e.'s relating to the constituent FA's have to be determined, the corresponding FA's are then derived using the Fire Engine toolkit, and these derived FA's are then summed, also using the toolkit, to provide $FA_{TR}(\rho, \varepsilon)$.

Once $FA_{TR}(\rho, \varepsilon)$ is constructed, certain adaptations to the conventional FA language recognition algorithm are required when scanning through a genomic sequence in search of the next TR. Some of the details relating to these adaptations will be discussed later.

For the present, consider the high-level description of FireμSat given henceforth. As mentioned previously, the algorithm requires as parameters:

- The lower and upper bound of motif lengths to be considered (*min* and *max* respectively);
- the maximum allowable motif error (ε - discussed in section 3.1);
- the maximum allowable substring error (τ - discussed in section 3.2);
- the penalty values used to calculate the substring error (p_m, p_d and p_i all explained in section 3.2);
- the maximum allowable number of ATRE's that may occur consecutively (α - discussed in section 3.2);
- the minimum number of TRE's that should occur before a string is output as a TR (β - explained in section 3.2) and;
- the genomic sequence itself ($gSeq$). (The development of the actual software is in progress the user is provided with an additional option to select default values for the respective threshold values.)

The following functions are assumed:

- $GenerateWords(\rho Length)$ generates a set of all words of length $\rho Length$ from the alphabet $\Sigma = \{\texttt{A},\texttt{C},\texttt{G},\texttt{T}\}$.
- $CreateFA_{TR}(\rho, \varepsilon)$ returns $FA_{TR}(\rho, epsilon)$ as discussed.
- $FindIndices(gSeq, FA_{TR}, \tau, \alpha, \beta, p_m, p_d, p_i)$ returns a set of index pairs in $gSeq$. A substring of $gSeq$ is a TR recognized by FA_{TR} within the constraints specified by τ, α and β as explained in section 3.2 if and only if its start and endpoint indices constitute a pair in the returned set. Note that the call to this function is independent of all prior and subsequent calls to it.

proc $Fire\mu Sat(min, max, \varepsilon, \tau, \alpha, \beta, p_m, p_d, p_i, gSeq)$
pre $\{(0 < min \leq max) \wedge (0 \leq \varepsilon) \wedge (\sigma \leq \tau) \ \wedge \ (0 \leq \alpha \leq tn_atreC) \wedge$
$(0 \leq \beta \leq tn_tre) \wedge (gSeq \in \Sigma^*)\}$
$indices := \phi$
; **for** $\rho Length : [min, max] \rightarrow$
; $words := GenerateWords(\rho Length)$
; $FASet := \phi$
; **for** $w : words \rightarrow$
; $FA_{TR} := CreateFA_{TR}(w, \varepsilon)$
; $indices := indices \cup FindIndices(gSeq, FA_{TR}, \tau, \alpha, \beta, p_m, p_d, p_i)$
rof
rof
post $\{(i, j) \in indices \Leftrightarrow \exists \rho : \Sigma^* \cdot |\rho| \in [min, max] \wedge$
$isTR(gSeq[i, j], \varepsilon, \tau, \alpha, \beta, p_m, p_d, p_i, \rho)\}$

Note that in order to use FA_{TR} appropriately in FireμSat, it is required that the final states of the original component FA's be identifiable in it. Note that of the features of the constructive algorithm introduced in the proof of *Rule 2, Part 3* of Kleene's algorithm is that if it is used to compute say $FA_X = FA_Y + FA_Z$, then every final state in FA_Y can be mapped to a final state in FA_X. The same holds true for every final state in FA_Z. Moreover, every final state in FA_X will either map to a final state in FA_Y or to a final state in FA_Z or to a final state in both FA_Y and FA_Z.

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

For reasons explained later, the number of symbols scanned since the last tally of a final state is stored in ℓ, and a flag *motif* is set to true that final state marked the end of a PTRE and to false, otherwise. Finally ε, is the maximum number of symbols by which an ATRE may differ from the motif. (The value of ε is easily determined and depends on the motif length as explained in section 3.)

The logic of how these counters are to be updated whenever a state, Q, of an FA_{TR} is being examined. A number of predicates are assumed that test whether Q is a final state ($isFinal(Q)$) and/or whether Q is a state that terminates a motif ($isPTRE(Q)$), and/or a deletion ($isDel(Q)$), insertion ($isIns(Q)$) or mismatch ($isMis(Q)$).

Note, specifically, that more than one of these conditions may hold for a final state, as forthcoming discussed. Dijkstra's guarded command language (GCL) is used, in which the semantics of the if-statement specifies that non-deterministic selection of the guards takes place if more than one guard evaluates to true. Therefore, to avoid ambiguity, guards have to be designed to be mutually exclusive. In each case, the body then adjusts the counters according to the rules already given above.

Thus, it will be seen that if a final state is of multiple types, then the PTRE counter (*tn_ptre*) takes precedence, followed by the mismatch counter (*n_m*), followed by the

deletions counter (n_d), followed by the insertion counter (n_i). By this it is meant that if a final state is encountered that is final for both PTRE's and mismatches, then the PTRE counter is incremented rather than the mismatch counter. Similarly, mismatches are incremented rather than deletions, etc.

However, there are a few exceptions to be dealt with in the case of an insertions final state being reached. Firstly, the insertions counter n_i is only incremented if the next state, R, is *not* also an insertion state. Secondly, suppose that the last TRE encountered was a PTRE (indicated by the flag *motif*) and that the number of transitions from this last PTRE state to this current insertion state (recorded in ℓ) is less than or equal to ε. It is then assumed that an insertion has been encountered instead of a motif ℓ transitions earlier. Consequently the tn_ptre counter that was previously incremented is now decremented.

Note that similar logic ought to be built in, to check that when arriving at a PTR state, a deletion was not incorrectly recorded less than ε transitions earlier. If so, the n_d, tn_atre and tn_atreC ought to be decremented. The outline below leaves out this logic in the interests of overall simplicity. However, it is built into the implemented algorithm.

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

$R := nextState(Q, nextSymbol)$
if $isFinal(Q) \rightarrow$
  if $isPTRE(Q) \rightarrow$
    $tn_ptre, tn_atreC := tn_ptre + 1, 0$
    $; \ell, motif := 0, true$
  $[\!]$ $(\neg isPTRE(Q) \wedge isMis(Q)) \rightarrow$
    $tn_atre, tn_atreC, n_m := tn_atre + 1, tn_areC + 1, n_m + 1$
    $; \ell, motif := 0, false$
  $[\!]$ $(\neg isPTRE(Q) \wedge \neg isMis(Q) \wedge isDel(Q)) \rightarrow$
    $tn_atre, tn_atreC, n_d, := tn_atre + 1, tn_atreC + 1, n_d + 1$
    $; \ell, motif := 0, false$
  $[\!]$ $(\neg isPTRE(Q) \wedge \neg isMis(Q) \wedge \neg isDel(Q)) \rightarrow$
    if $(\ isIns(Q) \wedge \neg isIns(R)) \rightarrow$
      if $(\ell \leq \varepsilon \wedge motif) \rightarrow tn_ptre := tn_ptre - 1$ **fi**
      $; tn_atre, tn_atreC, n_i := tn_atre + 1, tn_atreC + 1, n_i + 1$
      $; \ell, motif := 0, false$
    $[\!]$ $(isIns(Q) \wedge isIns(R)) \rightarrow$
      $\ell := \ell + 1$
    fi
  fi
$[\!]$ $\neg isFinal(Q) \rightarrow \ell := \ell + 1$
fi

5 Conclusion

The above code indicates that counter and threshold values are only adjusted whenever a final state is reached in FA_{TR}. This gives FireμSat a certain Moore machine character - Moore machines print output only in relation to the state that is reached—irrespective of which arc was followed to get to that state. However, in this case, when particular states are reached corresponding counters are adjusted, instead of printing specific characters as would be required in a Moore machine.

If a dead-end state is reached or $\tau < \sigma$ or $\alpha < tn_atreC$, then the processing on the applicable FA_{TR} will terminate, and the TR scanned up to that point will be output, provided that $\beta \geq tn_tre$. Scanning will then continue in search of the next TR.

The various parameters have already been discussed. They were derived in close collaboration with molecular biologists with a view to enhancing the useability of the algorithm. For example the software under construction allows the user to allocate penalties in a very sensitive manner, which enables the user to predetermine relatively easily at least what type of repeats will definitely be detected. It should be noted that these useability features are a direct consequence of using FA technology.

The implementation of FireμSat, is in progress. Preliminary runtime results show that FireμSat copes satisfactorily in searching for microsatellites with at most one mutation. Runtime results of motifs with length four or five where two mutations are allowed compares very well with STAR ([6]) but not that well with Tandem Repeats Finder ([3]). However, it should be emphasised that FireμSat provides the user with a degree of flexibility and usability does not appear to be available in the other algorithms. Future initiatives are directed at the completion of the FireμSat software and at reporting comparative results in more detail.

References

[1] C. Abajian: *Sputnik*. Online: http://espressosoftware.com/pages/sputnik.jsp.

[2] G. Benson: *A space efficient algorithm for finding the best non-overlapping alignment score*. Theoretical Computer Science, 145 1995.

[3] G. Benson: *Tandem repeats finder*. Nucleic acids research, 27(2) November 1999, pp. 573 – 580.

[4] A. T. Castelo, W. Martins, and G. R. Gao: *Troll: Tandem repeat occurrence locator*. Bioinformatics Applications Note, 18(4) 2002, pp. 634–636.

[5] E. Coward and F. Drablos: *Detecting periodic patterns in biological sequences*. Bioinformatics, 14 1998.

[6] O. Delgrange and E. Rivals: *Star: an algorithm to search for tandem approximate repeats*. Bioinformatics, 20(16) June 2004, pp. 2812–2820.

[7] S. Kannan and E. Myers: *An algorithm for locating nonoverlapping regions of maximum alignment score*. SIAM Journal on Computing, 25(3) 1996, pp. 648–662.

[8] S. Karlin, M. Morris, G. Ghandour, and M. Leung: *Efficient algorithms for molecular sequence analysis*. In: Proceedings of the National Academy of Sciences of the United States of America, 85 1988.

[9] R. Kolpakov and G. Kucherov: *Finding maximal repetitions in a word in linear time*. In: 40th FOCS. IEEE Computer Society Press, 1999.

[10] R. Kolpakov and G. Kucherov: *Finding approximate repetitions under hamming distance*. In: ESA: Annual European Symposium on Algorithms, Lecture Notes in Computer Science, 2161 2001.

[11] G. Landau and J. Schmidt: *An algorithm for approximate tandem repeats*. In: Proceedings of the 4th Combinatorial Pattern Matching Conference, Lecture Notes in Computer Science 648, 1993.

[12] G. Landau, J. Schmidt, and D. Sokol: *An algorithm for approximate tandem repeats*. Journal of Computational Biology, 8(1) 2001, pp. 1–18.

[13] M. Main and R. Lorentz: *An $O(n \log n)$ algorithm for finding all repetitions in a string*. Journal of Algorithms, 5 1984.

[14] A. Milosavljevic and J. Jurka: *Discovering simple dna sequences by the algorithmic significance method*. Computer Applications in Biosciences, 9(4) 1993, pp. 407–411.

[15] G. Myers and M.-F. Sagot: *Identifying satellites and periodic repetitions in biological sequences*. Journal of Computational Biology, 5(3) 1998, pp. 539–554.

[16] E. Rivals: *Eric rivals's homepage*. Online: http://www.lirmm.fr/~rivals/tete-en.html.

[17] E. Rivals, J. Delahaye, O. Delgrange, and M. Dauchet: *A first step toward chromosome analysis by compression algorithms*. In: Proceedings of the First International IEEE Symposium on Intelligence in Neural and Biological Systems (INBS '95), 1995.

[18] E. Rivals, O. Delgrange, J.-P. Delahaye, M. Dauchet, M.-O. Delorme, A. Henaut, and E. Ollivier: *Detection of significant patterns by compression algorithms: the case of approximate tandem repeats in dna sequences*. CABIOS, 13 1997.

[19] J. Schmidt: *All highest scoring paths in weighted grid graphs and its application to finding all approximate repeats in strings*. SIAM Journal on Computing, 27 1998.

[20] J. Stoye and D. Gusfield: *Simple and flexible detection of contiguous repeats using a suffix tree*. Theoretical Computer Science, 27 2002.

[21] M. Thurston and D. Field: *Msatfinder: detection and characterisation of microsatellites*. Online: http://www.genomics.ceh.ac.uk/~milo/msatfinder/, 2005.

[22] N. Tran, B. Bharaj, E. Diamandis, M. Smith, B. Li, and H. Yu: *Short tandem repeat polymorphism and cancer risk: influence of laboratory analysis of epidemiologic findings*. Cancer Epidemiology Biomarkers and Prevention, 13 2004.

[23] I. Van den Bergh: *Finding microsatellites in whole genomes*, Master's thesis, Technische Universiteit Eindhoven, 2006.

[24] B. W. Watson: *The design and implementation of the FIRE Engine: A C++ toolkit for finite automata and regular expressions*. Online: http://alexandra.tue.nl/extra1/wskrap/publichtml/9411065.pdf.