CHAPTER III

RESEARCH METHODOLOGY
CHAPTER III

RESEARCH METHODOLOGY

This chapter discusses the research methodology employed in proposed algorithm that is to be developed for detection of the Exact Tandem Repeats. In the present investigation various homologus and oncogene sequences are taken from the website of http://www.ncbi.nlm.nih.gov as input. The proposed algorithm is developed using JAVA Netbeans software. Before explaining the various steps involved in development of proposed algorithm, the definition of Exact tandem repeats and approximate tandem repeats are described in section 3.1 and 3.2. Then the characteristics of tandem repeats, various definitions, lemma and definition of suffix matrix are discussed in Section 3.3. In Section 3.4 the detailed implementation of algorithm is discussed.

Definition of Approximate Tandem Repeats

Model of Formation of Tandem Repeats

When a unit is repeated it’s assumed that there’s a substitution probability pS and an insertion or deletion probability pI (insertions and deletion are assumed equally likely) at each position and it’s also assumed that these errors are independent from each other. Since the total error probability is the sum of pS and pI , the probability of a character being copied without errors is called pM or probability of match where pM = 1 − (pS + pI ). Therefore when aligning two units of a repeat with period t, roughly pM × t matches, pS × t substitutions and pI × t indels are expected.
Terminology

The best way to define an approximate repeat is with a distance based approach with an edit distance or alignment score metric since insertions and deletions are allowed in the approximate repeats. First approximate single tandem repeats will be defined.

**Definition 3.1** A string $B = AA'$ is an approximate single tandem repeat if and only if the number of errors (substitutions plus indels) in the alignment of its two repeating units $A$ and $A'$ with respect to a score function $f$ is less than or equal to some similarity criteria $\Theta_{max}(t)$ where $t$ (called the period) is the length of the first repeating unit $X$. In the algorithm the alignment score function $f$ is an input and is universal for all tandem repeats. The threshold $\Theta_{max}(t)$ is a function of the parameter $p_m$ of the probabilistic assumption described above and the period of the repeat $t$. It act as a threshold of similarity between the various repeating units.

Since both single and multiple repeats can be detected by proposed algorithm in this thesis, a new definition of tandem repeats will be introduced later to express both single and multiple repeats. The only difference between the detection of single and multiple repeats is in the last step of the verification phase. To extend the definition of single tandem repeats to multiple repeats, a combination of the consensus and the neighboring approximate tandem repeat definitions according to an alignment score criteria is used because of its similarity to the definitions of tandem repeats. Definition 3.2 gives the formal definition of an approximate tandem repeat.

**Definition 3.2** A string $B = A_1 A_2 \ldots A_c A_{c+1}$ is defined as a Tandem Repeat if and only if the following conditions hold for some string $C$ (which is called the consensus pattern) with length $t_C$:

- For all $i$ such that $1 \leq i \leq c$; $e_i = E(A_i', C')$ should be less than or equal to $\Theta_{max}(tC)$ where $E(A_i', C')$ is the number of errors in the optimal global alignment $E(A_i', C')$ (with respect to some score function $f$) of the repeating unit $X_i$ and consensus pattern $C$. 
• For all $i$ such that $1 \leq i < c$, $E(A_i', A_{i+1}')$ should be less than or equal to $\Theta_{\text{max}}(t)$ where $E(A_i', A_{i+1}')$ is the number of errors in the optimal global alignment $E(A_i', A_{i+1}')$ (with respect to score function $f$) of the two adjacent repeating units $A_i$ and $A_{i+1}$ and $t$ is the total number of matches, mismatches, insertions and deletions in that alignment.

• if $A_{c+1}$ is a full string then $e_{c+1} = E(A_{c+1}', C_p')$ should be less than or equal to $\Theta_{\text{max}}(\text{length}(C_p))$ where $E(A_{c+1}', C_p')$ is the number of errors in the optimal global alignment $E(A_{c+1}', C_p')$ (with respect to score function $f$) of the partial repeating unit $A_{c+1}$ with some nonempty prefix $C_p$ of the consensus $C$ such that $C_p \neq C$.

Alignment score function $f$ is an input to the algorithm and is universal for all tandem repeats. The threshold $\Theta_{\text{max}}(t)$ is the same threshold used in the above given definition of single repeats, however here it is also used to express the lower bound of similarity between the repeating units and the consensus pattern.

Now definition of the tandem repeat $B = A_1A_2 \ldots A_c A_{c+1}$ is defined with some additional properties.

**Consensus pattern** The string $C$ is called the consensus pattern of the tandem repeat $B$.

**Consensus period**- The length $t_c$ of the consensus pattern is called the consensus period of the tandem repeat $B$.

**Period** The period $t_i$ of a repeating unit $A_i$ is defined as the length of the string $A_i$ for $1 \leq i \leq c$. The period of the tandem repeat $B$ is the most common period among the periods of the repeating units. If there exists more than one candidate then the one closest to the consensus period is selected.

**Copy number** The sum $c + \text{length}(C_p)/t_c$ is called the copy number of the tandem repeat $B$. It is the sum of the number of full repeating units plus the fraction of the partial repeating unit.
Score of a repeating unit \( S_i = F(A_i', C) \), which is the score of the optimal global alignment (with respect to some score function \( f \)) of the repeating unit \( A_i \) with consensus pattern \( C \), is called the score of the repeating unit \( A_i \).

Score of the partial repeating unit \( S_{c+1} = F(A_{c+1}', C_p) \), which is the score of the optimal global alignment (with respect to some score function \( f \)) of the partial repeating unit \( A_{c+1} \) with some nonempty prefix \( C_p \) of the consensus \( C \) such that \( C_p \neq C \), is called the score of the partial repeating unit.

Total score The sum \( S(Y) = \sum_{i=1}^{c+1} S_i \) is called the total score of the tandem repeat \( Y \) where \( S_i \)'s are the scores of each repeating unit defined above.

It is observed that all of the above properties depend on the consensus pattern \( C \). Since there may be more than one consensus pattern that satisfies the conditions in the definition, none of the properties is unique. It is also noted that even if there’s a unique consensus pattern, the repeating units can be decomposed in different ways.

3.2 Definition of Exact Tandem Repeats

The notation \( Y = A_1A_2\ldots A_n \) will be used throughout this thesis to denote that \( B \) is the concatenation of the strings \( A_1, A_2, \ldots \) and \( A_n \). Similarly \( B = A^c \) is the notation for the concatenation of \( c \) copies of a string \( B \).

Definition 3.3 A perfect or exact single tandem repeat with period \( t \) is a string \( B = AA \) which is a concatenation of two copies of a string \( A \) of length \( t \) where \( t \geq 1 \). The string \( A \) is called the repeating pattern of \( B \).

Definition 3.4 An exact or perfect multiple tandem repeat with period \( t \) and copy number \( c+f \) is a string \( B = A^cA' \) which is a concatenation of \( c \) copies of a string \( A \) of length \( t \) and, if \( f \neq 0 \), a prefix \( A' \) of \( A \) where \( t \geq 1 \) and \( c \geq 2 \) and \( f = \text{length}(A')/t \). The string \( A \) is called the repeating pattern of \( B \).
For example the sequence
\[ S = \text{ATCTAGAAGCTAGCTATGAGTCGTAGCTAGC} \]
contains a perfect single tandem repeat with period 4 at position 8. The problem of detecting perfect tandem repeats which consists of only two repeating units (perfect single tandem repeats) is also called as “detecting squares in strings” in computer science literature.

**Definition 3.5 (repeat region, repeat pattern)** A repeat region is a region which contains only repeating sequence. A DNA sequence usually contains multiple repeat regions, as shown in Figure 3.1. Repeat regions are marked in black and a repeat region usually contains multiple repeat patterns. As shown in Figure 3.2, “TATAT” is a repeat pattern in the repeat region and it matches the sequence exactly. “TAT” is also a repeat pattern, but it does not match the sequence exactly. We need to add insertions (“T”) in order to match it with the sequence exactly. “TA” is also a repeat pattern that does not match the sequence exactly.

![Figure 3.1](image1.png)  
**Figure 3.1 Multiple repeat regions in a DNA sequence**

![Figure 3.2](image2.png)  
**Figure 3.2 Multiple repeat patterns in a repeat region**
The input is the DNA sequence, which consists of four characters. The problem is to find the interesting repeat patterns that repeat at least twice in the sequence. A DNA sequence usually contains multiple repeat regions and a repeat region usually contains multiple repeat patterns. [85]’s algorithm only returns maximum three patterns per repeat region. The repeat pattern is the best sequence that can align with a repeat region, so the repeat pattern can be a sequence that never occurs in the sequence in [85]’s algorithm.

**Example 3.1** The sequence shown in Table 3.1 contains an exactly matching repeat pattern AGC.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>AGC AGC AGC AGC AGC AGC AGC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern</td>
<td>AGC</td>
</tr>
<tr>
<td>Match</td>
<td>EXACT MATCH</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3.1 Exactly matching patterns</th>
</tr>
</thead>
</table>

Since DNA may tolerate some mutations during the evolution, a large amount of repeat patterns we see in the DNA sequence are mere approximates. In Example 3.3, the pattern is an approximately matched pattern.

**Example 3.2** AGC is a repeat pattern in the following sequence as shown in Table 3.2 but the matches are not exact matches this time, there are substitution and deletion in the alignment.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>AGC AGC ATC AGC AC AGC AGC AGC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern</td>
<td>AGC</td>
</tr>
<tr>
<td>Match</td>
<td>With substitution and deletion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3.2 Approximately matching pattern</th>
</tr>
</thead>
</table>
Example 3.3 Note that only the best alignment template is returned so the pattern may never appear in the sequence. For instance, take a look at the following sequence as shown in Table 3.3:

<table>
<thead>
<tr>
<th>Sequence</th>
<th>ACGAA ACGGTA CGTT ACGT AGGTA A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern</td>
<td>ACGTA</td>
</tr>
<tr>
<td>Match</td>
<td>With insertion, deletion and substitution</td>
</tr>
</tbody>
</table>

Table 3.3 Pattern that never occurs in the sequence

Sequence ACGTA will be returned as the repeat pattern. Although it never occurs in the input sequence, it aligns the best with the sequence compared with other sequences. It is returned as a pattern.

Example 3.4 Consider the DNA sequence “CACAC CACAC CACAC CACACAC”. “CACAC” is the core pattern because “CACAC” aligns with the sequence exactly. “CAC” and “CA” are its variations because they need to add insertions/deletions in order to align with the sequence. Each repeat region always contains one core pattern and it may contain zero, one, or multiple variations of the core pattern. For instance, “CTCTCTCTCTCT” only contains one core pattern, which is “CT”. Of course “TC” is also a repeat pattern that matches exactly. From the redundant definition that we will give later, we know that it is a redundant pattern of “CT”.

Definition 3.6 (core pattern) The pattern which has the largest similarity score among all the repeat patterns in a repeat region is called a core pattern; at the same time there are patterns that are different variations of the core pattern, which contain some insertions, deletions or substitutions and their scores are smaller than the core pattern but still larger than the threshold. Repeat region contains clusters of repeat patterns. Each cluster contains a core pattern and patterns that are variations of the core pattern.
3.3 Characteristics of Tandem Repeat Patterns

In this section, the various characteristics of tandem repeat patterns are observed. **Figure 3.4** gives the distribution of tandem repeats in a DNA sequence.

**Figure 3.3 Distribution of tandem repeat in a DNA sequence**

**Example 3.5** From the repeats of **Figure 3.4**, one repeat is zoomed in for detail study as shown in **Figure 3.5**. It is assumed that one repeat pattern has length $d$ and repeats three times in a sequence. If the pattern matches exactly in the sequence, every item should match the item that is $d$ items apart in the next copy of the repeat. Since the tandem repeats we are studying are approximate matching repeats, which means when a pattern is aligned with the sequence, mismatches are observed. In **Figure 3.4**, the sequences for region 1 and region 2 are shown. In region 1, the repeat pattern “TGTGCC” matches the sequence “TGTGCCTGTGCC” exactly twice. Therefore all item pairs with interval 6 matches. In region 2, since pattern “TCAA” does not match with the sequence “TCAATCATTAG” exactly, not every item pair matches with interval 4. But one can still observe that there are three item pairs in the first and second copy with interval 4. Based on this observation, the following lemma is introduced.

**Figure 3.4 Characteristics of Tandem Repeat Patterns**
3.3.1 Occurrence List Construction

**Definition 3.7 (Occurrence list)** The occurrence list \( \langle p_1, p_2, \ldots, p_n \rangle \) of a certain item \( j \) of a sequence \( p = p_1 p_2 \ldots p_n \) is an ordered list such that \( p_1 = p_2 = \ldots = p_j = j \). \( (j \in \{A, G, C, T\}) \) and if \( j \neq k \) (0 < \( k \)), \( p_k \neq j \). From Definition 3.7 one can see that the occurrence list of a certain item is just an ordered list, which keeps the records of the positions of various items occurs in the sequence.

**Lemma 3.1** Given a DNA sequence, the occurrence lists of all the items from the itemset of the sequence contains the complete information of the sequence relevant to repeat pattern mining. Instead of original sequence, occurrence list can be directly mined.

**Proof:** From the definition of the occurrence list, it is observed that the occurrence list for a certain item contains all the positions where the item occurs. All the positions of a certain item will be recorded in its occurrence list and no positions for other items will be recorded there. Therefore original sequence can be uniquely constructed using occurrence lists of the items. Thus if one already has the occurrence lists for all the items, one can directly mine the occurrence lists. The example below shows the process of constructing the occurrence lists for a given short sequence.

**Example 3.6** An occurrence list is built after scanning the sequence as shown in table 3.4 using a sequence ATTCA.

<table>
<thead>
<tr>
<th>Item</th>
<th>Occurrence list for the item</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1 · 5 · 10 · 16 · 17 · NULL</td>
</tr>
<tr>
<td>T</td>
<td>2 · 3 · 6 · 7 · 8 · 11 · 12 · 13 · 15 · NULL</td>
</tr>
<tr>
<td>C</td>
<td>4 · 9 · 14 · NULL</td>
</tr>
<tr>
<td>G</td>
<td>18 · NULL</td>
</tr>
</tbody>
</table>

Table 3.4 Occurrence list construction for each item
Since tandem repeats tend to be highly reserved during evolution: only a small number of mutations will be tolerated, we have the following property for tandem repeat pattern.

**Property 1 (property of the tandem repeat pattern)** The percentage of the mutations is less than 25% in the tandem repeat patterns in the DNA sequence. Based on this property we have the following lemma.

**Lemma 3.2** If a pattern is a tandem repeat pattern, there must exist two contiguous repeat copies wherein the number of items that repeat with the same interval is larger than half of the length of the pattern.

**Proof:** As per the definition of tandem repeat, a tandem repeat has to repeat at least twice. From the property 1, we know that the percentage of the mismatch of the tandem repeat is less than 25%. Therefore, the percentage of matches for the first and second copies of the repeat should be more than 50%. Thus we have the lemma.

**Definition 3.8 (potential pattern)** A repeat pattern with length $d$ is called a potential pattern if and only if the number of items that matches items that is $d$ items apart is larger than $d/2$.

**Redundant Pattern Removing**

Since one region may contain many patterns, which are identical, if their similarity score is repeatedly calculated with the original sequence, it is costly and unnecessary. For example, pattern $A = \text{“GT”}$ and pattern $B = \text{“GTGT”}$ all have a similarity score of 50 with the input DNA sequence, and the left-hand side boundary and right-hand side boundary for their repeat region are the same. One can say that pattern $B$ is a redundant pattern of pattern $A$ because with the information of pattern $A$, one can derive the information of pattern $B$ from it. Since there exist numerous of redundant patterns in the same repeat regions, it is necessary to remove those patterns before calculating their similarity score. First, let us introduce a few definitions on redundant repeat patterns.
Definition 3.9 (Redundant pattern) Pattern $p_1 <a_1a_2…a_n>$ and $p_2 <b_1b_2…b_m>$ are called identical if $a_1 = b_1$, $a_2 = b_2$, $a_3 = b_3$, …. $a_n = b_n$. Pattern $p_1 <a_1a_2…a_n>$ is called the redundant pattern of pattern $p_2 <b_1b_2…b_m>$ if the two patterns satisfy one of the following conditions; otherwise $p_1$ is called non-redundant pattern of $p_2$.

1. $n = m$ and there exists an integer $x$ ($x > 1$) such that either $p_1$, $p_2$ are identical or $<a_1a_2…a_n>$ and $<b_x…b_n b_1…b_{x-1}>$ are identical;

2. $m > n$, $m \mod n = 0$ and $<a_1a_2…a_n>$ and $<b_1b_2…b_n>$ satisfy condition 1.

Let us for a instance, “CGTACA” is considered the redundant pattern of pattern “TACACG” because they satisfy condition 1 and CGT is considered the redundant pattern of pattern “GTCGTCGTC” because they satisfy condition 2. Whenever similarity score of a pattern is larger than the threshold, it is marked as a reference pattern for verification later. Note one only needs to mark those that have a similarity score that is larger than the threshold, because if the score of a pattern $P_i$ is smaller than the threshold, even though there is a pattern $P_j$ which is the same as $P_i$, one cannot disregard it because $P_j$ may get a score that is larger than $p_i$ and still have a chance to pass the threshold. Therefore before calculating a candidate $P_i$ in the test list, one must test it to see if there exists a candidate $P_j$ that has been tested before that satisfies the following conditions:

- $P_i$ is called a redundant pattern of $P_j$.
- The similarity score is larger than the threshold between $P_j$ and the sequence;
- $P_i$’s starting point is in the range of the repeat region of $P_j$.

Therefore, it is not required to calculate $P_i$’s similarity score since $P_i$ is a redundant pattern of $P_j$ and the information of that pattern is already available. But if there exists a previously tested candidate $P_j$ whose score is larger than the threshold, and $P_i$’s starting point is not in the range of the repeat region, $P_i$ would be a new repeat pattern. For any potential pattern $P_i$ on the test list, one calculates its similarity score with the given DNA sequence using the two-step wrap around dynamic programming method as described above, but only if there is no such pattern $P_j$ that matches $P_i$ and has been
The pattern is marked as a reference pattern for redundant pattern testing, if the similarity score is larger than the threshold. One prunes the patterns whose similarity score is less than the threshold. After completion of scanning of the sequence, interesting and unique patterns are obtained. All non-interesting and redundant patterns have been pruned.

### 3.4 Long and Short Potential Patterns Detection

**Definition 3.10 (Long, short patterns)** Given a repeat pattern \( p < p_1 p_2 \ldots p_n > \), the threshold for the minimum similarity threshold \( \cdot \) and the score for a match \( \cdot \cdot_m \), if the length of the repeat pattern is less than \( \cdot \cdot / \cdot_m \), \( p \) is considered as a **short pattern**. Otherwise, \( p \) is considered as a **long pattern**.

**Lemma 3.4** If a pattern with length \( d \) is a truly interesting repeat pattern, there must exist a repeat region with length \( 2 \cdot \cdot / \cdot_m \) and number of matches with distance \( d \) is larger than \( \cdot \cdot / \cdot 2 \cdot_m \).

**Proof:** From lemma 3.4, one knows that the length of minimum repeat region is \( \cdot \cdot / \cdot_m \). One also knows that the percentage of the mutations in the pattern will not exceed 25% Repeat from property 1. If there exists a truly interesting repeat pattern with a similarity score larger than the threshold \( s \), then there must exist a repeat region with length \( 2 \cdot \cdot / \cdot_m \) of the repeat region with more than 50% matches.

**Definition 3.11 (Suffix Matrix):** The numerical representation of a string is suffix matrix. The classical string matching for a pattern \( p \) and a text \( t \) finds all occurrences of \( p \) in \( t \). In many applications ranging from string matching to computational molecular biology, the same text is queried many times with different patterns. Efficient solutions for this problem are based on constructing an index data structure of \( t \) that contains an occurrence of \( p \) as an index in \( t \). Various kinds of index data structures for one-dimensional strings have been developed such as suffix tree, suffix matrix and so on.

The suffix matrix is a compacted matrix that represents all suffixes of a text string. A suffix matrix for a text \( t \) of length \( n \) over an alphabet \( \Sigma \) can be built in \( O(n \log |\Sigma|) \).
time. We can search a pattern \( p \) of length \( m \) in \( O(m \log |\Sigma|) \) time using the suffix matrix. It is noted that the construction time and the query time depend on the alphabet size. Although it was mainly designed for pattern matching purposes, the suffix matrix is useful for many other applications of string processing. The suffix matrix is basically a sorted list of all the suffixes of a text string and can be constructed in \( O(n \log n) \) time. When the sorted list is coupled with information about longest common prefixes (lcps), string searches can be answered in \( O(m + \log n) \) time using a simple augmentation to a classic binary search. In practice, suffix matrix use less space than suffix trees, but the construction takes more time.

In this thesis we propose Isuffix arrays and Zsuffix arrays that generalize suffix arrays to suffix matrices. We first define linear representations of suffix matrix. In order to sort the linearly represented suffixes, we develop a new partitioning technique based on Hopcroft's function partitioning [86]. By applying the technique, we present a simple and practical algorithm for constructing suffix matrix. Our algorithm is independent of the alphabet size and can be easily extended to higher dimensions. In this thesis we present an \( O(n^2 \log n) \) time construction algorithm for suffix matrix, which is the first algorithm for constructing two-dimensional suffix matrix directly.

### 3.5. Linear Representation of Square Matrices

Given an \( n \times n \) matrix \( A \), we denote by \( A[i : k; j : l] \) the submatrix of \( A \) with corners \( (i; j), (k; j), (i; l), \) and \( (k; l) \). When \( i = k \) or \( j = l \), we omit one of the repeated indices. An entry of matrix \( A \) has a symbol from an alphabet \( \Sigma \), on which a total order \( \alpha \) is defined. Consider a string \( x \) over alphabet \( \Sigma \). The \( i \)th suffix (resp. prefix) of \( x \) is defined as the largest substring of \( x \) that starts (resp. ends) at position \( i \). We generalize this definition of suffixes to higher dimensions: For \( 1 \leq i; j \leq n \), the suffix \( SA_{ij} \) of matrix \( A \) is the largest square submatrix of \( A \) that starts at position \( (i, j) \) in \( A \). That is, \( SA_{ij} = A[i : i + k; j : j + k] \) where \( k = n - \max(i, j) \).

The completeness constraint is that every square submatrix of \( A \) must be associated with a prefix of a suffix of \( A \) and the common prefix constraint is that the same square submatrices of \( A \) must be a common prefix of some suffixes of \( A \), whatever the definition of prefix is. To satisfy these constraints, we adopt a linear representation of
a square matrix. Let $\Sigma = \cup_{i=1}^{\infty} \Sigma_i$, where the letter I represents linear shapes. We refer to the strings of $\Sigma$ as Icharacters and we consider each of them as an atomic item. We refer to $\Sigma$ as the alphabet of Icharacters. Two Icharacters are equal if and only if they are equal as strings over $\Sigma$. Moreover, given two Icharacters $I_w$ and $I_u$ of equal length, $I_w < I_u$ if and only if $I_w$ as a string is lexicographically smaller than $I_u$ as a string.

We describe a linearization method for a square matrix $A[1 : n; 1 : n]$. We linearize the matrix along its main diagonal. When we cut a matrix along the main diagonal, it is divided into an upper right half and a lower left half. Let $a(i) = A[i + 1; 1 : i]$ and $b(i) = A[1 : i + 1; i + 1]$ for $1 \leq i < n$, i.e., $a(i)$ is a row of the lower left half and $b(i)$ is a column of the upper right half. Then $a(i)$'s and $b(i)$'s can be seen as Icharacters.

The linearized string $IA$ of matrix $A[1 : n; 1 : n]$, which is called the Istring of matrix $A$, is the concatenation of Icharacters $IA[1]; \ldots; IA[2n-1]$ that are defined as follows: (See Figure 3.5)

(i) $IA[1] = A[1; 1]$;
(ii) $IA[2i] = a(i)$, $1 \leq i < n$;
(iii) $IA[2i + 1] = b(i)$, $1 \leq i < n$.

Since $IA$ is composed of $2n-1$ Icharacters, the length of Istring $IA$ is $2n-1$. The $k$th $IPrefix$ of an Istring $IA$, denoted by $IA[1..k]$, is the concatenation of Icharacters $IA[1]; \ldots; IA[k]$. For each Icharacter $IA[l]$, $1 < l \leq 2n-1$, tail($IA[l]$) is the last character of $IA[l]$ and body($IA[l]$) is the rest of $IA[l]$. See Figure 1. Given two Istrings $IA$ and $IB$, $IA < IB$ if $IA$ is smaller than $IB$ in the lexicographic order of Icharacters in
IA and IB. The notion of Istrings, where the letter 'I' represents linear shapes, is a simple variant of Lstrings, but it plays a crucial role in constructing two-dimensional suffix arrays as well as two-dimensional suffix matrix.

### 3.6 Isuffix Arrays

Given a text matrix $T[1 : n; 1 : n]$, we will define Isuffixes of $T$. Let $\#i$ be a special symbol not in the alphabet $\sum$ such that $\#i < \#j < a$ for integers $i < j$ and each symbol $a \in \sum$. We first define the extended matrix $A[1 : n + 1 ; 1 : n + 1]$ of $T$ as follows: (See Figure 3.6)

(i) $A[i, j] = T[i, j]$ for every $1 \leq i; j \leq n$;
(ii) $A[k, n + 1] = A[n + 1, k] = \#k$, for every $1 \leq k \leq n + 1$.

Consider a suffix $SA_{ij}$, $1 \leq i; j \leq n$, of extended matrix $A$. The Istring of $SA_{ij}$ is called an Isuffix of $T$ and denoted by $\alpha_{ij}$. Since special symbols were added, there cannot exist a pair of Isuffixes $\alpha_{ij}$ and $\alpha_{uv}$ such that $\alpha_{ij} = \alpha_{uv}$. The number of all Isuffixes of $T$ is $n^2$.

Now we define two-dimensional suffix arrays, Isuffix arrays. The Isuffix array is a suffix array of all the Isuffixes of a given matrix, and consists of three tables POS, Llcp, and Rlcp. The three tables are basically the same as those of Manber and Myers. The basis of the Isuffix array is a lexicographically sorted table POS. We define a table POS[$1 : n^2$] of matrix $T$ as follows:

An element POS[$k$] has the start position $(i; j)$ if and only if $\alpha_{ij}$ is the $k^{th}$ smallest Isuffix in lexicographic order. We will construct table POS by sorting all the Isuffixes $\alpha_{ij}$ for $1 \leq i; j \leq n$.

Given two Isuffixes $\beta$ and $\gamma$, let lep($\beta, \gamma$) be the length of the longest common prefix of $\beta$ and $\gamma$ when $\beta$ and $\gamma$ are regarded as one-dimensional strings. Consider all the possible triples $(L, M, R)$ that can arise in a binary search on the interval $[1 : n^2]$, where $L$, $M$, and $R$ denote the left point, middle point, and right point of the interval that remains to be searched. There are exactly $n^2 - 2$ such triples, each with a unique
midpoint $M \in [2 : n^2-1]$ and we have $1 \leq L < M < R \leq n^2$ for each triple. Let $(L_M, M, R_M)$ be the unique triple containing midpoint $M$. $Llcp$ and $Rlcp$ are tables of size $n^2-2$ such that $Llcp[M] = lcp(\alpha_{POS[L_M]}; \alpha_{POS[M]})$ and $Rlcp[M] = lcp(\alpha_{POS[R_M]}; \alpha_{POS[M]})$.

![Figure 3.6: An Isuffix array of a square matrix.](image)

**Example 3.7**: In Figure 3.8, we give an example of Isuffix arrays. Given a 3 X 3 text matrix $T$, we show a table $POS$ that is a lexicographic sorted array of all Isuffixes of $T$. We also show all middle points that can arise in a binary search. If a middle point $M$ is 7, then the unique triple is $(5,7,9)$. In this case, we have $Llcp[7] = lcp(\alpha_{3,3}, \alpha_{2,3}) = 1$ and $Rlcp[7] = lcp(\alpha_{2,3}, \alpha_{1,2}) = 2$

### 3.5 Tandem Repeat Pattern Finder

In this section, the functionality of suffix matrix based algorithm is defined in detail. In the preprocessing phase, the input DNA sequence or any other biological sequence,
which is a string, will be first represented in the form of a suffix matrix. Then the
frequencies of various possible patterns are found using the suffix matrix. For
example, if the different characters found in a string are G,A,C,T then the different
patterns that can be obtained from these are G, A, C, T, GT, AC, CT, ATC, GCT,
GGTC, GAAA and so on. Once we get the frequencies of the patterns, the frequent
patterns are discovered by pruning the infrequent patterns using Apriori algorithm.
Therefore the preprocessing phase includes 3 main steps given below:-

Step 1. Suffix Matrix generation
Step 2. Finding frequencies of various patterns.
Step 3. Pruning of the infrequent patterns using the Apriori data mining
algorithm.

Step 1: Suffix matrix generation

Suffix matrix is the numerical representation of a string. The rows of the matrix
represent the various characters present in the string and are indexed in the order in
which they appear in the string. For example, if the string is GACTGGTCAGT then
the first row belongs to character G, second row to A, Third row to C and fourth row
to T as they appear in the same order as in the string. New column is added for each
occurrence of the a new character. The elements of the matrix give the next character
position. Each element is made up of pairs. The first value in the pair represents the
index of the character that occurs next in the sequence. The second value represents
the column in which the next character is represented as pair. Example: Consider the
following sequence GACTGGTCAGT. The rows are G – index 1, A – index 2, C –
index 3, T – index 4. Thereafter, a matrix can be created with these rows. The matrix
initially contains only one column to enter the occurrence of first character in the
sequence. Therefore initially the matrix looks as follows.

<table>
<thead>
<tr>
<th>G-index 1</th>
<th>A-index 2</th>
<th>C-index 3</th>
<th>T-index 4</th>
</tr>
</thead>
</table>

59
Now we have to represent character G in the matrix. The index of the character next to (i.e., A) is 2. The character A will be represented in the first(1) column next. So the element G is represented as pair 2,1.

<table>
<thead>
<tr>
<th>G-index 1</th>
<th>(2,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-index 2</td>
<td></td>
</tr>
<tr>
<td>C-index 3</td>
<td></td>
</tr>
<tr>
<td>T-index 4</td>
<td></td>
</tr>
</tbody>
</table>

On the same manner the next character A is represented as 3, 1 (3 for the index of next character, C and 1 is the column number in which C will be represented next). Character C is represented as 4, 1 (4 for the index of next character T and 1 is the column number in which T will be represented next). The character T is represented as 1, 2 because the index of the next character i.e., G is 1 and the character G will be represented in the second column. Finally the matrix will obtained will be

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-index 1</td>
<td>(2,1)</td>
<td>(1,3)</td>
<td>(4,2)</td>
<td>(4,3)</td>
</tr>
<tr>
<td>A-index 2</td>
<td>(3,1)</td>
<td>(1,4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C-index 3</td>
<td>(4,1)</td>
<td>(2,2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T-index 4</td>
<td>(1,2)</td>
<td>(3,2)</td>
<td>(§,§)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.7 – Suffix matrix generated.

(§,§) will indicate the termination of the string.

**Algorithm to generate suffix matrix:**

INPUT: Gnomonic or Biological sequence: str

OUTPUT: two suffix matrices(M1[][] and M2[][]) one for each value appearing in the pair.

PROCESS:
/*searching the various characters in the string and assigning them indexes*/
str="";
char="";
str=input Biological or Genomics sequence.
FOR i=1 to str[i]!="\0"
index=0;
FOR j=1 to char[j]!="\0"
IF str[i]==char[j]
index=1;
END IF
END FOR
IF index!=1
char[count]=str[i];
count++;
END IF
END FOR
/* entering values into matrix*/
FOR or i=0;i<length of char;i++
count[i]=0;
END FOR
FOR i=0;i<length of str;i++
id=indexof character at position in string str.
IF i==length of str-1
one[id][count[id]]='$';
two[id][count[id]]='$';
count[id]++;
ELSE
int a=count[id];
one[id][a]=index of character at i in string str
count[id]++;
two[id][a]=count[index of character at i+1 in string char];
END IF
END FOR

Figure 3.8 Algorithm for suffix matrix generation.
Step 2 - Finding frequencies of various patterns.

There are two different types of patterns
1. Single character patterns G, A, C, T
2. Patterns with frequency more than 2.

Single character patterns:
For single character patterns the frequency is given by the column count of that particular character. Form our previous example, the frequency of G is 4, A is 2, C is 2 and T is 3.

Patterns with length more than 2:
In this the suffix matrix is traversed and find the frequencies. For example to find the frequency of the pattern GTC we start with finding GT it is found 2 times in row 1 since the index 4 is present 2 times. For the first occurrence of index 4 in row A, the value of the pair is 4,2 now we check in 4TH row 2nd column in the matrix. The character found is C as index is 3. Therefore the pattern present is GTC. Then we have to the second occurrence of GT is 4,3 in row G, hence we have to check the character present in 4th row and 3rd column position. and found $,$,. When the index is found to be $ that means that we have only GT but third character is not present. Hence the pattern is GT and is not counted. Hence the frequency of the pattern GTC is 1. For each of those patterns we find the frequencies using the above method.

Step 3: Pruning Infrequent Patterns Using Apriori Algorithm:
Apriori algorithm is used for pruning of infrequent patterns. Apriori is an algorithm to find all sets of items that have support no less than minsup. The support for an itemset is the ratio of the number of transactions that contain the itemset to the total number of transactions. Itemsets that satisfy minimum support constraint are called frequent itemsets. Figure 3.9 gives the apriori algorithm
input values for the minimum repeat range for each of the patterns having length 1,2
3,4,5,6 and so on were taken. For example if the minimum repeat range of di is given
as 2 then all the patterns of length those are repeated more than 4 times are found like
GTGT, ACAC etc and so on. If the minimum repeat value of tri is given to be 3, all
the patterns like GACGACGAC are found. We take the minimum repeat values from
the input and compute the minimum threshold and support values.

References

[85] Benson, G. “Tandem repeat finder: a program to analyze DNA sequences”,

[86] Domanic N. O. , and Preparata F. P..” A novel approach to the detection of
genomic approximate tandem repeats in the Levenshtein metric” Journal of

[87] Liao Yaqin Nancy,” Fault-Tolerant Repeat Pattern Mining on Biological Data”,
Simon Fraser University, 2001.

[88] Stoye J. and Gusfield D., “Simple and flexible detection of contiguous repeats

[89] Whitener W.Weldon,” Analysis of short tandem repeat variation in large scale
resequencing data,” Ph.D.dissertation, University of Cambridge, Wellcome Trust
Sanger Institute, St. Edmunds College, 2011.
[90] Domanic Onur Nevzat,” An Algorithm for Detecting Approximate Tandem Repeats in Genomic Sequences”, B. S., Middle East Technical University, 2004


