6.1 Introduction

Repeats are ubiquitous in large eukaryotic genomes[8,9,10]. Genome sequencing indicates that a large proportion of a vertebrate genome is composed of five major classes of repeat sequences: short and long interspersed elements, inactive retro-posed copies of cellular genes (usually referred as processed pseudogenes), simple short sequence repeats such as $A^n$ or $(CGG)^n$, segmental duplications, and tandem repeats. They complicate genome assembly, genome comparison, and the study of genome rearrangement. They also increase the difficulty in probe signal analysis for microarray (that is, the probe signal may be inaccurate if the probe sequence overlaps with repeat regions). Biologically, the distribution of repeats is highly correlated with chromatin structure, which influences the activity of genes. For instances, Alu sequences are major target for histone H3-Lys9 methylation in humans, which silence many genes. Repeats are also believed to play significant roles in genome evolution and proneness to disease. For example, Pelizaeus-Merzbacher disease (PMD) commonly arises from genomic duplications of the dosage-sensitive proteolipid protein gene (PLP1). Hence, repeat identification has attracted considerable attention and many algorithms have been designed to identify the repeats in a genome.

There are several types of repeats:

- **Motif** is short sequence (of length 10-20) of repeat which usually serves as transcription factor binding site.

- **Micro-satellite** and **mini-satellite** are contiguous series of intact short sequence. These type of repeat is likely to differ between individuals. They was used to identify evidence in DNA finger print technique.

- **Mobile element repeats** (e.g. transposons, retrotransposons) have members that are typically separated in the genome.

Repeats can also be used as DNA fingerprint. Microsatellite and minisatellite are repeats in the form $\alpha^k$ where $\alpha$ is some short sequence. Microsatellite and
minisatellite may differ between individuals. And we can use it as DNA finger-printing. In addition, both microsatellite and minisatellite patterns can provide information about paternity. As an famous example, president Thomas Jefferson has no surviving son, and Eston Hemings is the last child of Jefferson’s slave Sally Hemings. It is suspected that Hemings is the son of Thomas Jefferson instead of his supposed father Thomas Woodson. To validate this suspect, the number of repeats in 11 microsatellites are obtained from the Y chromosomes of their male-line descendants, as shown below:

- Male-line Descendants of President Jefferson’s uncle:
  15, 12, 4, 11, 3, 9, 11, 10, 15, 13, 7.

- Male-line Descendants of Eston Hemings:
  15, 12, 4, 11, 3, 9, 11, 10, 15, 13, 7.

- Male-line Descendants of Thomas Woodson:
  14, 12, 5, 11, 3, 10, 11, 13, 13, 13, 7.

It seems that the repeat numbers are more similar between the descendants of Jefferson’s uncle and Eston Hemings. Hence, president Thomas Jefferson may be the father of Eston Hemings.

### 6.1.1 Related work

Finding repeat is a special case of alignment problem in which the sequence is aligned to itself. Therefore, we can use alignment tool such as BLAST to identify pairs of repeated regions. However, the scale and the required level of sensitivity of the problem is different, therefore we still need some specially designed methods for identifying repeat.

Another important job after finding repeated region is to classify these repeats into families. This job is not quite easy because repeated copies are degraded, and sometime overlap with each other. Some research focused on grouping repeats into different classes or families. For example, RepeatFinder[1] (Volfovsky et al., 2001), RECON[11] (Bao and Eddy, 2002), RepeatGluer[6] (Pevzner et al., 2004), PILE[3](Edgar and Myers, 2005). However, in this note, we don’t go in depth into these classification methods.

We have two classes of repeat finding methods. The first one is repeat finding with prior knowledge, e.g.: RepeatMasker[13], MaskAid[12]. They use database of annotated elements to compare with input sequence. The second class is De-novo repeat finding e.g.: REPuter[1] (2000), PALS[3] (Edgar and Myers, 2005), Eulerian path approach[2] (Zhang and Waterman 2005).

Below, we describe different repeat finders one by one.
6.2 RepeatMasker

RepeatMasker[13] is developed by A. F. A. Smit and P. Green. It requires a library of known repeat sequences like RepBase database. As the name has suggested, RepeatMasker is used to mask, remove repeat in the sequence. RepeatMasker performs an approximate string matching of the given sequence against all entries in the repeat library. The comparison is done by a program called cross_match, an efficient implementation of the Smith-Waterman-Gotoh algorithm.

MaskAid[12] written by Joseph A. Bedell, Ian Korf, and Warren Gish. It is an accelerated version of RepeatMasker. Basically, it replaces cross_match by a wrapper for WU-BLAST.

This approach has many disadvantages. It cannot discover new repeat, therefore it does not add much information to our knowledge. Besides, it requires a repeat library. However, repeat library is not universal. For each genome, we need to compile the repeat library manually. Automatically reconstruct the repeat library is still difficult.

6.3 Hashing approach

Hashing is a very intuitive method. It can work well with the exact repeat problem and extensible to inexact repeat problem.

Definition 6.1 (Exact repeat) A pair of equal length substring $S[i_1..j_1]$ and $S[i_2..j_2]$ is an exact repeat if $S[i_1..j_1] = S[i_2..j_2]$.

Definition 6.2 (Maximal exact repeat) An exact repeat is maximal if it is not contained in any other exact repeat.

For example, $S = ACGTAGCTCCACTGGGTATCTGGATC$

CGTACTTC is a length 5 maximal exact repeat; TAC is an exact repeat, but it is not maximal.

Definition 6.3 (K-mer) is a substring of length $K$ of sequence $S$.

Example Sequence $S$ produces the following hash table:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | .. | .. | .. | .. | .. | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 |
| S = A C A A T A A G T C G A T .. | T | C | G | C | A | A | G | T | C | G | A | G | .. | C | A | T |
### 6.3.1 Algorithm

Given a sequence $S$, we list all the possible k-mers and their positions inside sequence $S$ in a hash table. For each pair of left-maximal positions having the same k-mer, we extend to the most right match position in order to find the maximal repeat. Algorithm 1 states the details of the algorithm.

**Algorithm 1** Hashing algorithm

1: for all k-mer $w$ occurring in more than 2 positions do  
2:     Let $L_w$ be the list of positions of $w$  
3:     for all pair of positions $(p_1, p_2)$ in $L_w$ do  
4:         if $S[p_1 - 1] \neq S[p_2 - 1]$ then  
5:             /* Extend the alignment to find maximal repeat */  
6:                 len ← $k$  
7:         while $S[p_1 + len] = S[p_2 + len]$ do  
8:             len++  
9:         Report ($S[p_1..p_1+len-1], S[p_2..p_2+len-1]$) is a pair of maximal exact repeat.

For example, k-mer $w$ = “AAG” in the previous string occurs at positions 6 and 89. Since $S[5] \neq S[88], S[13] \neq S[96]$, we know that this pair of repeats is left-maximal. Then, we extend the pair of repeats to the right end we found that $S[6..12] = S[89..85]$ and $S[13] \neq S[86]$ , we report “AAGTCGA” as a the maximal exact repeat.

### Discussion

Hashing is actually a filtering method to reduce the number of pairs of positions we want to check for repeats. We will miss all the repeats whose length is less
than k, but generally we want to find repeats exceed certain length. Moreover, if we choose k too short, for example \( k = 1 \), we may have to check up to \( O(n^2) \) pairs of positions. Therefore, hashing method is effective when there are not too many k-mers reoccurred, thus we don’t need to spend too much time on extension to get the maximal repeat. In practice, the length of the k-mer \( k \approx \log_{|\Sigma|} n \) (see 6.6.1 for explanation), for example, \( |\Sigma| = 4 \) in DNA. Although the required space of this method is bounded by \( O(n) \) words; however, the real memory usage is still too much compared to other methods. Due to memory limitation, k is usually between 10 and 13. This approach is not very practical for finding degraded repeat in big genome.

6.4 Suffix tree approach

The most popular tool for finding de novo repeat at the moment is REPuter. The core algorithm of this software is suffix tree. It can guarantee to find all degenerated repeats of length at least \( l \) and have at most \( k \) errors. REPuter has two versions; the first one can only find maximal mismatch repeat (MMR), the second one can find all maximal difference repeat (MDR) defined below. Both versions rank the significant of their repeats base on a probabilistic value called E-value.

REPuter was designed to find repeats in both forward and backward strains. The definition for repeat in forward stain, \( F\text{-repeat} \) for short, is the same as Definition 6.1. A repeat in backward strain (or palindromic repeat, \( P\text{-repeat} \) for short) is the repeat of reverse compliment of original DNA substring. Given input sequence \( S \), REPuter builds a suffix tree of \( xSyS^r z \) where \( S^r \) is the reverse compliment of \( S \). Then, REPuter can easily find maximal exact F-repeat and P-repeat in the suffix tree.

For example, Figure 6.1 shows the suffix tree for sequence “gtcaca”. “ca” is a maximal forward repeat (occurs at positions 3 and 5), and “gt” is a maximal backward repeat (occurs at positions 1 and 4).

6.4.1 Maximal mismatch repeat (MMR)

Definition 6.4 (K-mismatch repeat) A pair of equal length substrings \( S[i_1..j_1] \) and \( S[i_2..j_2] \) is a k-mismatch repeat if \( d_H(S[i_1..j_1], S[i_2..j_2]) = k \) (\( d_H \) is the hamming distance between 2 strings).

Definition 6.5 A k-mismatch repeat is maximal if it is not contained in any other k-mismatch repeat.
Figure 6.1: Example of REPuter suffix tree for “gtcaca”

For example, S = ACGTAGCTCCACTGGGTATCTGGATC

GTAGCT is a length 5 maximal 1-mismatch repeat; TAGCT is a 1-mismatch repeat, but it is not maximal.

A k-mismatch repeat divides $S[i_1..j_1]$ and $S[i_2..j_2]$ into $k + 1$’s maximal exact repeats $w_0, w_1, \ldots, w_k$. Let $l$ be the length of the k-mismatch repeat. The length of the longest $w_i$ is

$$\max_{0 \leq i \leq k} \{|w_i|\} \geq \left\lfloor \frac{l - k}{k + 1} \right\rfloor = \left\lfloor \frac{l}{k + 1} \right\rfloor$$

For example, $l = 11, k = 3$ the length of the longest $w_i$ is $\left\lfloor 11/(3 + 1) \right\rfloor = 2$.

**Lemma 6.6** Every maximal covering k-mismatch repeat of length $l$ contains a maximal exact repeat of length greater or equal to $\left\lfloor \frac{l}{k+1} \right\rfloor$.

**Proof:** If there is no such maximal repeat, that means $\forall w_i, |w_i| < \left\lfloor \frac{l}{k+1} \right\rfloor$

The length of the repeat is $l < (k + 1) \times \left\lfloor \frac{l}{k+1} \right\rfloor \leq l$

This leads to a contradiction.

**Definition 6.7** All maximal exact repeats of length $l$ are called seeds.

REPuter (in Algorithm 2) first finds all seeds; then it and tests if each seed can be extend to a k-mismatch repeat. More precisely, for each seed $((i_1, j_1), (i_2, j_2))$, table $H_{left}$ and $H_{right}$ of size $k + 1$ are computed such that for each $q \in [0, k]$, $H_{right}$ is the maximum number $p$ such that $d_H(S[i_1..j_1 + p], S[i_2..j_2 + p]) = q$. We have similar definition for $H_{left}$, thus $H_{left}(q) + H_{right}(k - q)$ is the length of
Algorithm 2 Find Maximal Mismatch Repeat

1: Build suffix tree for $S$
2: Compute all seeds
3: for all seed $(S[i_1..j_1], S[i_2..j_2])$ do
4: /* Extension phase */
5: for all $0 \leq q \leq k$ do
6: $H_{\text{right}}[q] = \max \{ p | d_H(S[i_1..j_1+p], S[i_2..j_2+p]) = q \}$
7: $H_{\text{left}}[q] = \max \{ p | d_H(S[i_1..j_1], S[i_2..j_2]) = q \}$
8: for $0 \leq q \leq k$ do
9: if $j_1 - i_1 + 1 + H_{\text{left}}[q] + H_{\text{right}}[k-q] \geq l$ then
10: Report $(S[i_1 - H_{\text{left}}[q]..j_1 + H_{\text{right}}[k-q]], S[i_2 - H_{\text{left}}[q]..j_2 + H_{\text{right}}[k-q])]$

the extended region in which there are $q$ mismatches on the left side and $k-q$ mismatches on the right side.

To compute $H_{\text{left}}$ and $H_{\text{right}}$ efficiently, REPuter uses longest common prefix data structure.

Definition 6.8 Given any two position $i$ and $j$ in string $S$, we have $\text{lcp}(i, j) = r$ if and only if $S[i..i+r] = S[j..j+r]$ and $S[i..i+r+1] \neq S[j..j+r+1]$.

$lcp$ data structure can be pre-computed, thus we can retrieve it in $O(1)$ time[1].

Lemma 6.9 If $H_{\text{right}}[t] = r$, then $H_{\text{right}}[t+1] = r + \text{lcp}(i + r, j + r + 1) + 1$ with $t \geq 0$

Proof: As the definition of $H_{\text{right}}$, $H_{\text{right}}[t] = r$ implies $d_H(S[i..i+r-1], S[j..j+r-1]) = t$ and $S[i+r] \neq S[j+r]$.

We also have $\delta = \text{lcp}(i + r + 1, j + r + 1)$ imples $S[i + r + 1..i + r + 1 + \delta] = S[j + r + 1..j + r + 1 + \delta]$.

Let $x = r + \text{lcp}(i + r + 1, j + r + 1) + 1$, from these two implications, we have $d_H(S[i..i+x], S[j..j+x]) = t + 1$.

Hence, we can compute $H[1], \ldots, H[k]$ in $O(k)$ time.

Complexity analysis

- Step 1 and 2 can be done in $O(n)$ time
- Step 3 will iterate $z$ times
  - Where $z$ is the number of seeds.
\[ z = E[n^2 / |A|^s] \text{ where } s = \left\lfloor \frac{l}{k+1} \right\rfloor. \]

- Each iteration in Step 3 takes \( O(k) \) time.
- In total, the running time is \( O(n + zk) \)

### 6.4.2 Maximal Difference Repeat (MDR)

**Definition 6.10 (k-difference repeat)** A pair of equal length substrings \( S[i_1..j_1] \) and \( S[i_2..j_2] \) is a k-difference repeat if \( d_E(S[i_1..j_1], S[i_2..j_2]) = k \).

The length of k-difference repeat \( S[i_1..j_1] \) and \( S[i_2..j_2] \) is \( \min(j_1 - i_1 + 1, j_2 - i_2 + 1) \).

\( d_E \) is the edit distance of two strings. The edit operations are replacement, insertion and deletion.

**Definition 6.11** A k-difference repeat is maximal if it is not contained in any other k-difference repeat.

By generalizing the MMR algorithm, we can solve this problem in \( O(n + zk^3) \) time, where \( z \) is the number of seeds \( z = E[n^2 / |A|^s], \ s = \left\lfloor \frac{l}{k+1} \right\rfloor. \)

### 6.4.3 Significance of repeats

**Definition 6.12** E-value of a repeat is defined to be the number of strings of the same length or longer with the same number of errors or fewer that are expected to occur in a random DNA sequence of length \( n \).

Let \( p = 1 / |\Sigma| \) be the probability of one character appear in the sequence \( S \).

**E-value for exact repeat**

\[
E[\# \text{ of maximal exact repeat of length at least } l] = E[\# \text{ of left-maximal exact repeat of length } = l] \\
= \sum_{2 \leq i_1 \leq i_2 \leq n-l+1} P_r \left[ S[i_1..i_1 + l - 1] = S[i_2..i_2 + l - 1], \ S[i_1 - 1] \neq S[i_2 - 1] \right] \\
+ \sum_{2 \leq i_2 \leq n-l+1} P_r \left[ S[1..l] = S[i_2..i_2 + l - 1] \right] \\
= \sum_{2 \leq i_1 \leq i_2 \leq n-l+1} p^l(1-p) + \sum_{2 \leq i_2 \leq n-l+1} p^l \\
= \binom{n-l}{2} p^l(1-p) + (n-l)p^l
\]
E-value for k-mismatch repeat

\[ E[\# \text{ of maximal } k' \text{-mismatch repeats where } k' \leq k \text{ of length } \geq l] \]

\[
= \sum_{k'=0}^{k} \sum_{l'=l}^{n} \sum_{1 \leq i_1 \leq i_2 \leq n} \Pr \left[ d_H(S[i_1..i_1 + l' - 1], S[i_2..i_2 + l' - 1]) = k', \right.
\left. S[i_1 - 1] \neq S[i_2 - 1], S[i_1 + l'] \neq S[i_2 + l'] \right]

\[
= \binom{n}{2} \sum_{k'=0}^{k} \sum_{l'=l}^{n-1} \binom{l'}{k'} p^{l'-k'} (1-p)^{k'+2}

\approx \binom{n}{2} \binom{l}{k} p^{l-k} (1-p)^{k+2}

E-value for k-difference repeat

Unlike k-mismatch, there is no analytical solution of \( \Pr[d_E(S_1, S_2) = k] \).

Let \( A_k(P) \) be the probability that \( P \) matches the prefix of a random string with edit distance \( k \). \( E[l, k] \) is the average of \( A_k(P) \) for all patterns \( P \) of length \( l \). It can be estimated as follows

- Generate 100 random patterns \( P_1, P_2, \ldots, P_{100} \) of length \( l \).
- For each \( P_i \), we generate 1000 random strings and estimate \( A_k(P_i) \) as the chance that \( P_i \) matches the prefix of those random strings.
- Let \( E[l, k] \) be average of \( A_k(P_i) \) for \( i = 1, 2, \ldots, 100 \).

Since variance of \( A_k(P_i) \) should be small, \( \Pr[d_E(S_1, S_2)] \approx E[l, k] \) where \( l = \max(|S_1|, |S_2|) \). We can pre-computed \( E[l, k] \). Therefore we have

\[
E[\# \text{ of maximal } k \text{-difference repeats of length } l] \approx \sum_{1 \leq i < j \leq n} E[l, k] = \binom{n}{2} E[l, k]

6.5 Eulerian path approach

This approach is used in local multiple alignment as well as repeat finding.
6.5.1 De Bruijn graph

**Definition 6.13 (Eulerian De Bruijn graph)** Each k-mer is represented by a directed edge in the graph and two edges are joint by a node if their k-mers overlap at (k-1) letters in the sequence. Identical k-tuples are represented by same edges.

For example: 3-mer ACG is represent as AC $\rightarrow$ CG.

**Definition 6.14** The multiplicity of an edge is the number of occurrence of the k-mer in the sequence.

![Diagram](image)

Figure 6.2: De Bruijn graph

For example: The de bruijn graph of $S = ACGTCACCTCGTCAT$ is shown in Figure 6.2.

Using a probabilistic analysis, the larger the multiplicity is, the more likely the edge represents a conserved k-tuple. And vise versa, the conserved repeats tend to be amplified in the graph by edges of large multiplicities.

6.5.2 Algorithm

**Algorithm 3 Eulerian path**

1: Construct De Bruijn Graph of $S$
2: Remove thin edges and cycles
3: repeat
   4:   Apply heaviest path algorithm to find the consensus
   5:   By banded local alignment, find all segments which are similar to the consensus.
   6:   Remove the information of those repeat segments from the De Bruijn graph.
7: until there is no significant repeats left

By using the same example above, after construction the De Bruijn graph, we tried to remove edges whose weight is smaller than the threshold. (The threshold
is determined using Poisson-like heuristic) In this example, we remove all edges with weight at most 1.

Find the heaviest path, which is CGTCA. By banded local alignment, find all segments in S which are similar to the consensus, in this example, we found two.

Eulerian algorithm was originally designed for finding multiple alignment. For preprocessing phase, it tries to remove edges whose weight is smaller a threshold. This threshold is determined using Poisson like heuristic. It also removes some other edges so that the remaining graph is DAG (see example in Figure 6.3).

In the searching phase, each iteration Eulerian algorithm can find one repeat family representative. It applies dynamic programming to find heaviest path, then scan the genome and do banded local alignment to find all segments which are similar to the consensus.

**Discussion** Eulerian path approach is an heuristic. It may miss some repeats. It runs approximately linear time.

### 6.6 Spectrum-based approach

#### 6.6.1 Spectrum

**Definition 6.15** spectrum $\Gamma$ of a strings $S$ is the set of all k-mers of $S$.

**Example** With $k = 3$, $S = ACGACGCTCACCCT$, the spectrum is $\{ACC, ACG, CAC, CCC, CCT, CGA, CGC, CTC, GAC, GCT, TCA\}$.

Two regions of a long enough repeat should share some k-mers. However, in this approach we don’t store the position of the k-mer. We just use the spectrum of k-mers to find potential repeat region.

**K-mer length**

The selection of K-mer length is crucial for this approach. On one hand, if $k$ is big, we may miss some obvious repeats. On another hand, if $k$ is too small, it does not help much to differentiate repeat from non-repeat, thus the running time is bad.
Algorithm 4 Spectrum-based approach

1: Initialize the hash table H to be an empty spectrum
2: for i=1 to n do
3: /* We maintain the invariant that H is a spectrum of S[i..i−1] */
4: Let x be the k-mer at position i
5: if x exists in H then
6: find_repeat(i)
7: Insert x into H

\[ Pr(a \text{ k-mer occurs by random in a sequence of length } n) \]
\[ Pr(\text{this analogs to throwing } n \text{ balls into } 4^k \text{ bins}) \]
\[ \approx 1 - (1 - 4^{-k})^n \approx 1 - \exp(-n/4^k) \]

We requires \(1 - \exp(-n/4^k) \leq \epsilon_1\), hence

\[ k \geq \log_4 n + \log_4 \epsilon_1 \]

If we set \( \epsilon_1 = 1/16 \). This implies

\[ k \geq \log_4 n + 2 \]

### 6.6.2 Feasible Extension

**Definition 6.16 (feasible extension)** Given the spectrum \( \Gamma \), a feasible extension for a k-mer \( w \in \Gamma \) is a k-mer \( w' = w[2..k]X \), \( (X \text{ is a character in the alphabet}) \) so that \( w' \) is in \( \Gamma \).

For example, \( S = \text{ACGACGTGATTAACCCCTCGACGTCATCCTC} \), set of feasible extensions of k-mer CGA is \{GAC, GAT\}

**Definition 6.17 (fooling probe)** Given spectrum \( \Gamma \) of \( S \), a k-mer \( w \), and a feasible extension \( w' = w[2..k]X \), \( w' \) is a fooling probe if \( wx \) is not a substring of \( S \).

For example, in the previous example, the string GAT is a fooling probe since CGAT is not a substring of \( S \).

**Definition 6.18** Path of feasible extension of spectrum \( \Gamma \) of \( S[1..i−1] \) is a sequence \( P \) such that k-mer \( x_j = P[j..j+k−1] \) is a feasible extension of k-mer \( x_{j−1} \) according to \( \Gamma \).
We have an observation that the path of feasible extensions of \( S[1..i-1] \) which match \( S[i..i+l-1] \) (\( l \) is length of the path) may be a repeat. The probability for this event is approximately \((1-\epsilon)^{l-k}\) (\( \epsilon \) is the probability of fooling probe).

For example, \( S = ACGACGCTATCGATGCCCTC \)

Spectrum \( H \) for \( S[1..10] \) is \{ACG, CGA, CGC, CTA, GAC, GCT, TAT\}. Starting from position 11, there exists a path of feasible extensions: CGA-C-G-C

This path corresponds to a length-6 substring in position 2. Also, this path has one mismatch when compared with the length-6 substring for position 11 (CGATGC).

**Path pruning condition**

Instead of fixing the number of mismatches, we may want to fix the percentage of mismatches, for example, 10% mismatches. This pruning strategy is length dependent. If the length of strings in \( \Gamma \) is \( r \), we will allow \( \tau(r) \) mismatches.

Let \( q \) be the mismatch probability and \( r \) be the length of the string. The probability that a string has \( s \) mismatches is

\[
P_q(s) = q^2 \sum_{j=s-2}^{r-2} \binom{r-2}{j} q^j(1-q)^{r-2-j}
\]

For a threshold \( \epsilon \) (say, 0.01), we set

\[
\tau(r) = \max \{s \leq s \leq r-2 | P_q(s) > \epsilon\} + 2
\]

### 6.6.3 Algorithm

**Input:**

1. a position \( i \)
2. the spectrum \( \Gamma \) for \( S[1..i-1] \)

**Output:** All paths of feasible extension such that

1. start at \( i \)
2. at most \( m \) mismatches with \( S \) start from position \( i \).
3. length at least \( l \)
Algorithm 5 Find Maximal Mismatch Repeat at position i

1: Let $K$ be the $k$-mer at position $i$
2: Set $\Gamma = \{K\}$ where $dist(K) \leftarrow 0$
3: for $j = i + k$ to $i + l$ do
4:     for all string $P \in \Gamma$ do
5:         Let $F$ be the set of feasible extension of $P$
6:         for all feasible extension $f \in F$ do
7:             if $f \neq S[j]$ then
8:                 $dist(Pf) \leftarrow dist(P) + 1$
9:             else
10:                 $dist(Pf) \leftarrow dist(P)$
11:     if $dist(Pf) \leq m$ then
12:         $\Gamma' \leftarrow \Gamma' \cup \{Pf\}$ /* Pruning criteria */
13: if $\Gamma' = \emptyset$ then
14:     return "no repeat"
15: $\Gamma \leftarrow \Gamma'$
16: Validate if the strings in $\Gamma$ are repeats and report them

One example of Find Maximal Mismatch Repeat at position 18 is shown below, at most 1 mismatch of length at least 9.

From position 18, we got CGA, search spectrum, we only get CGC and CGT, therefore, position 21 could be either C or T, which T is mismatch. The extension tree has two path now, for first path, ACG and ACC are included inside spectrum, which C is mismatch. With the same logic, finially, CGACGCGAT will be found, and it’s a length-9 string in position 2 actually.
ACGACGCGATTAAACCCTCGACGTGATCCTC

Figure 6.4: The spectrum extension from position 20

References


