Workshop Report 1: Systems Analysis and Motive Detection

Introduction

This report presents a novel Java-based algorithm for identifying recurring patterns, or motifs, within artificially generated genetic sequences. The algorithm was developed as part of the first workshop of the Systems Analysis course. By incorporating entropy-based filtering, the algorithm effectively distinguishes meaningful motifs from random occurrences. Our experimental results demonstrate the algorithm's accuracy and efficiency in detecting motifs of varying lengths, providing valuable insights into the challenges and complexities of motif identification in genetic sequences.

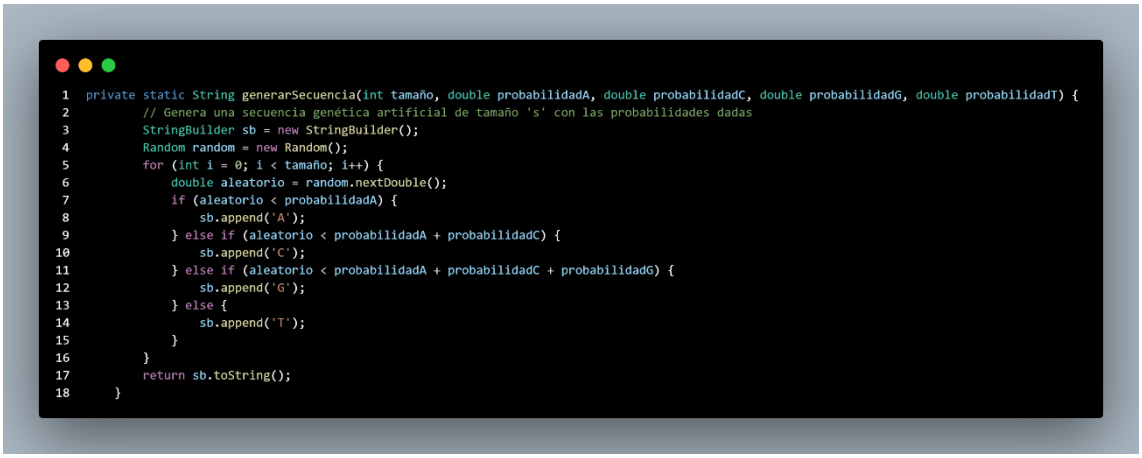
Objective

The main objective of the program is to simulate a database of artificial genetic sequences and detect the motif (character pattern) that appears most frequently within a specific size. Additionally, an entropy-based filter is implemented to select more diverse and potentially chaotic sequences.

Systemic Analysis

The program is made up of three main modules:

1. **Generation of Artificial Sequences:** This module creates a set of n random genetic sequences of size m. Each sequence is composed of nucleotides represented by the letters A, C, G and T. The selection probabilities of each nucleotide are configurable parameters. The generated sequences are saved in a text file.



```java
private static String generarSecuencia(int tamaño, double probabilidadA, double probabilidadC, double probabilidadG, double probabilidadT) {
        // Genera una secuencia genética artificial de tamaño 's' con las probabilidades dadas
        StringBuilder sb = new StringBuilder();
        Random random = new Random();
        for (int i = 0; i < tamaño; i++) {
            double aleatorio = random.nextDouble();
            if (aleatorio < probabilidadA) {
                sb.append('A');
            } else if (aleatorio < probabilidadA + probabilidadC) {
                sb.append('C');
            } else if (aleatorio < probabilidadA + probabilidadC + probabilidadG) {
                sb.append('G');
            } else {
                sb.append('T');
            }
        }
        return sb.toString();
    }
```

```java
// Crear secuencias genéticas artificiales
String[] secuencias = new String[númeroSecuencias];
for (int i = 0; i < númeroSecuencias; i++) {
    secuencias[i] = generarSecuencia(tamañoSecuencia, probabilidadA, probabilidadC, probabilidadG, probabilidadT);
}
```

2. **Entropy Calculation and Filtering:** This module calculates the Shannon entropy for each sequence. Entropy is a measure of disorder or chaos in a system. In this case, it is used to filter sequences with low entropy, that is, with high repetition of the same nucleotide. Sequences with entropy greater than or equal to the average entropy are retained for subsequent analysis.

```java
    private static double calcularEntropía(String secuencia) {
        // Calcular la entropía de una secuencia genética
        int[] frecuencias = new int[4]; // A, C, G, T
        for (char nucleótido : secuencia.toCharArray()) {
            switch (nucleótido) {
                case 'A' -> frecuencias[0]++;
                case 'C' -> frecuencias[1]++;
                case 'G' -> frecuencias[2]++;
                case 'T' -> frecuencias[3]++;
            }
        }
        double entropía = 0;
        for (int frecuencia : frecuencias) {
            if (frecuencia > 0) {
                double probabilidad = (double) frecuencia / secuencia.length();
                entropía -= probabilidad * Math.log(probabilidad) / Math.log(2);
            }
        }
        return entropía;
    }
}
```

```java
// Calcular entropía promedio
double sumaEntropía = 0;
for (String secuencia : secuencias) {
    sumaEntropía += calcularEntropía(secuencia);
}
double entropíaPromedio = sumaEntropía / secuencias.length;

```

3. **Motif Detection:** This module searches for the motif of size s that is most frequently repeated within the filtered sequences. All possible combinations of nucleotides of size s in each sequence are evaluated. If there are patterns with the same frequency, the one with the greatest number of consecutive repeat bases is selected.

```java
// Leer archivo y buscar la secuencia de caracteres de tamaño s que más se repite
Map<String, Long> frecuenciasSecuencias = new HashMap<>();
long startTime = System.nanoTime();  // Comenzar el cronometraje
try (BufferedReader reader = new BufferedReader(new FileReader(archivoSalidaFiltradas))) {
    String linea;
    while ((linea = reader.readLine()) != null) {
        for (int i = 0; i <= linea.length() - tamañoSecuenciaBuscar; i++) {
            String secuencia = linea.substring(i, i + tamañoSecuenciaBuscar);
            frecuenciasSecuencias.put(secuencia, frecuenciasSecuencias.getOrDefault(secuencia, 0L) + 1);
        }
    }
```

```java
// Encontrar secuencia más frecuente
String secuenciaMasFrecuente = null;
long frecuenciaMaxima = 0;
for (Map.Entry<String, Long> entry : frecuenciasSecuencias.entrySet()) {
    if (entry.getValue() > frecuenciaMaxima) {
        frecuenciaMaxima = entry.getValue();
        secuenciaMasFrecuente = entry.getKey();
    }
}
```

Complexity

The generation of artificial sequences has a time complexity of $O(n \times m)$, where n is the number of sequences and m is the size of each sequence.

The entropy calculation for each sequence has a time complexity of $O(m)$, where m is the size of the sequence.

Entropy-based sequence filtering has a time complexity of de $O(n)$, where n is the total number of sequences.

Motif detection has a worst-case time complexity of $O(n \times m \times s^3)$, where n is the number of filtered sequences, m is the size of the filtered sequences, and s is the size of the searched motif. This is because it iterates over all sequences, analyzes substrings

of size s, and in the worst case all possible combinations of nucleotides for the motif (s^3) must be evaluated.

Chaos Analysis

Entropy is used as an indirect measure of chaos in the system. Sequences with high entropy have greater diversity in their nucleotides, which can be considered a more chaotic system. The entropy-based filter allows selecting sequences that potentially contain less repetitive and possibly more interesting patterns for motif analysis.

Results

The program is executed taking as parameters the number of sequences (n), the size of the sequences (m), the probabilities of each nucleotide, the size of the motif to be searched (s) and the names of the files to save the generated sequences. and the filtered sequences.

The program output shows:

- The most frequent motif of size s found in the filtered sequences.
- The frequency of appearance of the reason.
- The average entropy of the generated sequences.
- The search time for the reason in seconds.
- The total execution time of the program in seconds.

Discussion of Results

The results of the program largely depend on the configured parameters. The selection of nucleotide probabilities and the size of the motif to search will influence the diversity of the sequences and the ease of detecting patterns. The average entropy can give an idea of the level of chaos in the generated sequences, and the execution time allows evaluating the efficiency of the algorithm used for motif detection.

Conclusion

The program implements a basic approach for motif detection in artificial genetic sequences. Random generation, entropy calculation and exhaustive search techniques are used to achieve the objective. Complexity analysis allows you to understand program performance based on the amount of data and the size of the desired motif. Incorporating entropy as a filter allows the analysis to be biased towards sequences with greater potential diversity.