Pattern Motif in Random DNA Sequences

Daniel David Cuellar 20221020081

Adrián David Malaver Machado 20231020068

Faculty of Engineering, Francisco José de Caldas District University

020-84, Systems Analysis

Dr. Andres Sierra

September 15, 2024

Table of Contents

1.Systemic analysis…….3

2.Complexity analysis……..……3

3.Chaos analysis…….4

4.Results……..…5

5.Discussion of results…….7

Conclusions…….……9

1. Systemic Analysis

Systemic analysis refers to the study of a system as a whole, understanding how its components interact to achieve a specific goal. In the provided code, the system is responsible for generating DNA sequences, detecting the most frequent motif, and measuring the time taken for this process. The system can be broken down into several key components:

- **Input Handling**: The system takes input from the user (number of sequences, sequence length, motif length) and performs checks to ensure the inputs are valid. This ensures that the system operates within expected boundaries, enhancing robustness. Invalid inputs lead to early termination of the program, avoiding unnecessary computations.
- **Random DNA Generation**: The `FillerBD` method generates random DNA sequences using the method `GeneradorBN`. Each sequence is built by appending random nucleotides ('A', 'C', 'G', 'T'). This randomness in the system's behavior introduces variability, but the systemic approach ensures each sequence conforms to the constraints provided by the user.
- **Motif Detection**: The `detectarMotif` function scans through each sequence, extracting motifs of the specified length, and stores them in a `HashMap` to track their frequency. This process highlights how different subsystems (motif extraction and frequency tracking) work together.
- **Time Measurement**: The system measures the time taken from when the user inputs data to the moment the motif is detected, showcasing the system's performance.

Systemically, the code operates in a sequential manner where one component relies on the previous. Any failure in one subsystem, such as invalid input or file-writing failure, will prevent the system from achieving its goal.

2. Complexity Analysis

Complexity analysis refers to evaluating the performance of an algorithm in terms of time (how long it takes) and space (how much memory it uses) as a function of its inputs.

- **Time Complexity**:
    - The **generation of DNA sequences** in the `FillerBD` method has a time complexity of **O(n * m)**, where `n` is the number of sequences and `m` is the length

of each sequence. This is because the program must loop through `n` sequences, and for each sequence, it generates `m` nucleotides.
    - The **motif detection** process in the `detectarMotif` method is more complex. It iterates over each sequence and, for each sequence, extracts motifs of the specified length. For each sequence of length `m`, it can extract `m - k + 1` motifs, where `k` is the length of the motif. Thus, the motif extraction and frequency counting has a time complexity of **O(n * m * k)** in the worst case, where `n` is the number of sequences, `m` is the length of the sequences, and `k` is the length of the motif. This means the time increases linearly with the number of sequences and sequence length, and quadratically when considering both motif length and the number of possible motifs.
- **Space Complexity**:
    - The **space complexity** for storing the DNA sequences is **O(n * m)**, since each of the `n` sequences of length `m` is stored in an `ArrayList`.
    - The **HashMap** used in the `detectarMotif` method has a space complexity of **O(u)**, where `u` is the number of unique motifs extracted. Since the number of possible motifs is bounded by `4^k` (the number of possible motifs of length `k` from the 4 nucleotides), in the worst case, the space complexity for storing the motifs and their frequencies could approach **O(4^k)**.

In summary, the time and space complexities are dependent on the number of sequences, the length of each sequence, and the motif length, which makes this problem computationally expensive for large datasets.

3. Chaos Analysis

Chaos analysis examines how small changes in initial conditions can lead to drastically different outcomes, often referred to as the "butterfly effect." In this context, chaos can emerge in several ways:

- **Randomness in DNA Sequence Generation**: The random DNA generation process (`FillerBD` and `GeneradorBN` methods) introduces a significant element of chaos. Even a slight change in the random seed could produce entirely different sequences, leading to a completely different outcome in the most frequent motif. This means that repeated runs of the same program, with slightly different random number generation conditions, could produce vastly different results, showcasing a chaotic system.
- **Motif Frequency Variability**: The detection of the most frequent motif also reflects a chaotic nature. Since DNA sequences are generated randomly, there may be cases where no motif is particularly dominant, or where a slight difference in sequence generation could make one motif significantly more frequent than others. This could lead to large swings in the identified most frequent motif and its frequency.
- **Time Variability**: The time taken to detect the motif is influenced by the specific input parameters and randomness in sequence generation. Slight changes in the dataset can lead

to significant differences in how long it takes to process and find the motif, adding another layer of unpredictability (chaos) to the system's behavior.

Chaos analysis in this code suggests that despite following deterministic rules, the underlying randomness in sequence generation can create a highly unpredictable system. This randomness can propagate throughout the code, affecting everything from motif detection to the time taken for the program to execute.

4. Results

For this section, we present a sample execution with the input provided. The complexity analysis**,** chaos analysis**,** and systemic analysis considerations will follow.

Input:

Enter the number of sequences:

1000

Enter the size of the sequences:

5

Enter the size of the motif:

4

Output (format changed):

DNA sequences generated:

GCATC/TATCT/GTGTA/GCACG/CATCG/TCGGT/ACGAG/TAGCG/TAGAA/AGTGC/ACC
CA/TAGCC/GTACC/TCACA/AGGAC/TCGTA/AAGGC/ATACT/TCTAA/GGATG/TAATC/
TGAAG/GAAAA/GGCGG/ATCTG/CCATT/TCACT/GAGGT/GAATA/CAGAC/GATAC/GT
TTT/TGCGT/TTATG/CCGTC/GCACA/AATTG/AGTTA/CAAAA/TGGAT/ACTAC/AGTCT/
ACTCA/GACGG/TCATC/ATATT/AATTA/GTAAG/GAGCG/CTACT/ACATT/AAAGA/CAC
GG/ACGTC/TGTTC/TAGAG/GTGCA/GCAGG/GGACG/TTCTC/ATCCG/AGTGG/CAGAA/
ATTGA/AGAGA/TTCGA/TTGAC/TTCGA/GGGTC/CTCAG/GCCGA/GCTCT/GAACA/ATA
TC/ACATT/CCCTG/TCTCA/CCTCT/CCGTG/ATCCG/CCATC/TGAAG/CTCTG/TCGCG/A
CAAT/AGTAA/CGAGC/ATAGT/AATTC/CCATA/AGGTT/AATTT/CTTTA/TAAAC/CCGG
C/CTAAG/GGACG/TGCGG/ACCAC/ATTAC/CAGTT/TAGGA/CCGGG/AAATC/AGACA/C
AGGG/AGCCG/GCTTG/GTATA/GGCCA/GCTGC/TGCAT/TAGTC/TTCGC/CGTTC/CCCT
A/TGCGC/CAGTA/CAATC/ATACG/GCATC/ATAAT/CATAG/CACAT/ACGAG/GAGCT/C
TCAT/CTTTT/AGAGT/CTTAA/AAAGT/AGCGC/ATGCC/TACTA/CGGAA/CAGAC/GGAG

T/ACAGA/TACTA/TACTC/GAAAG/GCGGC/ATAAG/TATAT/GAGCC/TGTTA/CGGAG/G
GGGG/GGTAA/GAAAT/CCTCG/TAGCC/CAGAG/GTACA/AGCTA/TCACG/TCACG/GTT
GG/AACAT/GTGGT/TGATT/CAGGT/TTCAC/ACAGC/CAAGG/ATGTG/CGTCC/GATCG/
GTGCA/TTACA/CGTTT/AGGAG/TCCAA/CGAAC/AATCA/ATAGG/CATAA/CAGCA/TA
GAA/ACTCC/TTCTA/AATTG/CTTAA/TGGTG/GTTTG/TAGGG/TCGCC/GCCTA/AGGAA/
CGCGG/ATACG/CAGCG/AACTG/ATCCC/CCGAC/GGGTC/AACTA/GGATG/TCAGA/AT
GTG/TCCGG/CTATT/TACCG/AAGTT/CCAGG/GATCG/TACGC/CTTGC/GTCCC/AGAGA
/ACCCT/GTCAT/TGGGA/GGCGT/CCGCG/TGAGC/CATTG/AATCC/AGGAC/GTCGC/TG
ACT/CTCTG/GGGCC/CGCAT/CATCA/AAATA/TTACG/CGCCC/AGTCG/CATTA/ATGTT/
CATGC/TATAA/GAATG/GGTGG/ACGTC/TGCTT/GGGTA/TTACA/TTCGG/AAATT/ATC
CT/ACGTA/GTAAT/GTCAG/ACCGA/GGCCC/CTCGC/ACGAT/GACGA/GCCTA/ATAAA/
GGTCT/TCATC/CACAT/AGATC/TTGTT/CTGTC/GCGGC/TTGCA/GTGGA/CTGGC/TCA
GG/AAACC/GACGG/GGAAG/GGGAC/ACCGG/CGTAG/TCAAT/TGGAG/CCTTC/ACTTC
/AAGAT/CCGGA/ATACT/CGAGG/CCACC/CCTTC/TTGTC/AAGGG/ATGGC/GGGGA/GG
TAG/CGGAA/TGTTT/TGGTT/GCTTC/GCCTG/AGGAC/CGCGT/TCACA/TGGTT/ATTTT/
CGAGT/GGAGA/ACACA/CGGCG/AGGGC/CGTCC/GAGAT/TTCGA/CTGCA/ATATT/TT
TAC/GGAGA/CCGCG/GGTAG/CAATG/CGTAT/CTGGC/CCCCC/TTCTT/GCTCT/GTTAT/
GATGC/TTCAT/AGGCT/AGTAG/GCATC/ACGCG/CTAAA/ACTCG/TCTCC/TATGC/TAT
AA/AGGGA/CTTGC/GCACA/CTTCC/AAATC/GAGGA/GCGCG/CTGCC/GACCG/TTGTA/
AGCCT/TTCGG/GCCCA/CCCGT/AATGA/AGACT/CGAAT/GATCG/ACTCA/CTTGA/TCT
CC/TCAGG/CCTCG/GTCAC/CCCCG/GTTCG/AAAAT/TAACC/AACTT/CCCGC/CCGCC/C
ACCG/TCTGA/CAGCC/AGAGG/TCACG/AACTT/TACGT/CCAAA/GAGTG/TTGGT/ACA
GA/CCCAG/ACTCA/CGACC/AGCTT/GTGTG/GCAGG/GACTT/TGGTT/GATTG/CGAAA/
TATGA/GGGAA/GCAGT/AATCC/TGCTA/ACTTC/AGAAG/TCAGC/CTACT/GGTAG/AT
ACA/TACAT/CCCCG/GACTG/TGGAG/TTACC/CTCTA/GTATC/AAATT/TTCGA/CGGAC/
CTGTT/CCCGG/GGGTT/CATTG/TAGAT/GTACC/GACGT/CCCGT/TAATT/CGGTA/TCC
CA/TGAGG/ATTTC/GAGAG/CTGTA/GCCGA/ATAGG/CCAGA/AGGGC/GTTGT/CAGCT/
TTAGA/ATATC/AAAAG/TGCTA/TTACG/TGGAG/AAATT/ATAGC/TCCCC/GAGCC/TCG
GT/TATGG/ATAGT/GTGAT/GCCAG/CCCCT/GGGTG/AATCT/ATGAT/TCTCG/CACTA/T
ATCT/TGCGA/TCGAA/AAAAA/AACCC/CATGC/TTAGG/TTATA/TCCCA/TAGTC/ATGG
A/CATAA/GAGAT/TCCGC/CCTTC/GTTGC/ACTCG/TGGAG/GGTCT/TAGCG/GGTAT/TC
GCG/CCTCT/GAAAT/CGCAT/CCGCT/CCTCC/GGGTG/TTCTA/GACGG/CTACG/CCACC/
ACGTT/TGTTC/AGATC/GACGC/CGGCA/CCGGC/AACTA/TGTGA/CCGAT/GCAGA/AG
AGC/AACGT/GGCAT/GTAGT/ACGCC/CGTTA/CAATT/CAGCC/TCGGG/TTTGT/CTGCG/
TTTGT/TGCGC/CGGCC/TCGTT/GTGAC/ACGAG/CACCT/CCGAG/GTTCG/AACGT/AAT
CG/TGAGA/CCCTA/AGGCG/CGGCC/GTCTT/AAGAG/TAGCG/GGCTT/CCCCT/TCCAG/
TAGGG/ATCAT/CCATG/GACAC/TACTG/GACTA/GTGAC/GCGAT/ATGAC/CTGCG/ACT
CA/GTCTA/GAGTA/TAATG/ACGAT/CAGTC/TTGGG/TCTCG/GTCTC/TTAGA/CGAGC/C
CAAT/CGCTC/ATAGC/CGTGA/GCATT/AGTTC/GTCTA/GACAC/CGCAA/AACAG/GTG
CG/GGAAT/TTATA/TTTAA/AGAAG/AGACC/TGGAT/CAGTT/CTAAC/GCTGC/CTCAA/
GGCTT/GAGCT/CGACT/GGTCA/GCTCG/GAATT/GTGGT/CCGCA/AAGGG/TGGTC/GTT
AG/CGATA/CCATT/CTTTT/GCAAC/TGGTT/CTTCC/CTTTA/GATTT/TTTCG/TCATA/AA
TTC/GGG

5. Discussion of Results

In this section, we will analyze the output presented in the example provided using three key frameworks: **Systemic Analysis**, **Complexity Analysis**, and **Chaos Analysis**. Each of these approaches offers a unique perspective to understand the characteristics of the generated DNA sequences, the process of motif detection, and the performance of the algorithm.

Systemic Analysis

Systemic analysis involves studying how the different parts of the system interact and contribute to the overall behavior. In this case, the DNA sequences generation, motif detection, and performance metrics can be evaluated as components of a larger system designed for bioinformatics simulations.

- **DNA Sequence Generation**: The system generates 1000 DNA sequences of 5 base pairs each. The input size (number of sequences) and length of each sequence play a crucial role in determining the system's computational load. These sequences are purely random, meaning each nucleotide (A, T, G, C) has an equal chance of occurring, leading to a vast number of possible combinations. The complexity here scales linearly with the number of sequences and their length.
- **Motif Detection**: The motif of size 4 ('ACGA' in this case) is detected 17 times among the generated sequences. This process involves comparing each subsequence of length 4 within each DNA sequence, which is systematically done across all sequences. The detection mechanism can be described as a sliding window approach, where each window of size 4 nucleotides is compared against the target motif.
- **Performance Measurement**: The system reports the time taken to find the motif as **292 milliseconds**. This gives an insight into the efficiency of the motif detection algorithm. The time is directly proportional to the size of the input data and the efficiency of the comparison operations. The performance indicates that the algorithm can handle this size of input relatively quickly.

Complexity Analysis

From a computational perspective, the complexity of the system primarily lies in the motif detection process, as generating random sequences is a linear-time operation.

- **Time Complexity**: Given that we have 1000 sequences, each of length 5, and we are searching for a motif of size 4, the number of comparisons required is approximately $1000 \times (5-4+1) = 10001000 \times (5 - 4 + 1) = 10001000×(5−4+1)=1000$ motif comparisons. For each sequence, the algorithm slides a window of size 4 across the sequence, checking for matches with the target motif. Since each comparison is constant time, the total time complexity for this operation is $O(n \cdot m)O(n \cdot m)O(n·m)$, where nnn is the number of sequences and mmm is the length of each sequence. In this case, $n=1000n = 1000n=1000$ and $m=5m = 5m=5$, leading to a time complexity of $O(1000 \cdot 5) = O(5000)O(1000 \cdot 5) = O(5000)O(1000·5)=O(5000)$, which is a manageable workload.

- **Space Complexity**: The primary data structure used is the list of DNA sequences, which consumes space proportional to the number of sequences multiplied by their length, i.e., O(n×m)O(n \times m)O(n×m). Additionally, the HashMap used for motif detection takes up space proportional to the number of unique motifs of size 4, which is relatively small given the short length of the DNA sequences. Hence, the space complexity is O(n×m)O(n \times m)O(n×m), and for this example, O(1000×5)=O(5000)O(1000 \times 5) = O(5000)O(1000×5)=O(5000), which is also efficient.

Chaos Analysis

Chaos analysis looks at how small changes in the input could drastically alter the system's output, particularly in systems that exhibit non-linear or unpredictable behavior.

- **Randomness in DNA Generation**: The DNA sequences are generated randomly, and each run of the system will produce entirely different sequences. This introduces an inherent element of unpredictability. The motif 'ACGA' appeared 17 times in this run, but in another execution with the same parameters, the frequency could be drastically different depending on how the random nucleotides are distributed.
- **Sensitivity to Input Parameters**: A slight change in input parameters, such as increasing the sequence length or the number of sequences, would exponentially increase the number of possible motifs and the complexity of finding them. For example, if the sequence length were increased to 10, the number of comparisons would double, increasing the chances of finding more occurrences of the motif but also increasing the computational load.
- **Stochastic Nature**: The stochastic nature of the random sequence generation means that certain motifs may appear more frequently in one run than in another purely by chance. This variability can make it difficult to predict the exact outcome of any single run, even though the overall system behavior remains statistically consistent.

Performance Insights

The example execution produced the following performance metrics:

- **DNA sequences generated**: 1000 sequences of length 5.
- **Most frequent motif**: 'ACGA' detected 17 times.
- **Time taken to detect the motif**: 292 milliseconds.
- **Dataset successfully saved in**: 'src/dataset.txt'.

This performance can be considered efficient for a dataset of this size. The time taken (292 ms) suggests that the system scales well with smaller datasets, and the implementation is optimized for quick detection of motifs.

Conclusions

In analyzing the generated DNA sequences and their motif detection, we observe insights related to entropy, chaos, and their implications for computational methods. The characteristics of the dataset provide a valuable perspective on how randomness influences biological data analysis.

1. **Entropy and Diversity:** The high entropy observed in the dataset signifies a considerable degree of diversity and unpredictability among the sequences. Entropy measures the amount of uncertainty or randomness present, and in this context, it indicates that the generated DNA sequences cover a broad range of possible configurations. This level of diversity mimics real-world biological sequences, which are naturally varied due to evolutionary processes and genetic variation. Such randomness is crucial for creating realistic datasets that are useful for testing motif detection algorithms and other bioinformatics tools.
2. **Chaos and Sequence Generation:** The chaotic nature of sequence generation, characterized by the random assignment of nucleotides, results in a dataset that is inherently non-deterministic. This randomness is analogous to chaotic systems in nature, where small initial differences can lead to vastly different outcomes. In DNA sequence generation, this chaos ensures that no predictable patterns emerge unless specifically sought, reflecting the complexity and variability found in natural genetic sequences. Understanding this chaotic behavior is essential for developing algorithms that can handle the unpredictability of biological data effectively.
3. **Implications for Motif Detection:** The combination of high entropy and chaotic sequence generation presents challenges for motif detection. The complexity of the dataset, with its high degree of randomness, requires advanced computational techniques to identify recurring patterns or motifs accurately. The HashMap approach employed in this case demonstrates an effective method for handling such complexity, as it allows for efficient storage and retrieval of sequence data. This scenario underscores the need for robust and scalable algorithms capable of managing the intricate nature of biological sequences and highlights the importance of adapting computational methods to address the challenges posed by entropy and chaos in genetic data analysis.