

**T.C.**

**MARMARA UNIVERSITY**

**FACULTY of ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

CSE4065 Assignment #1

**REPORT**

***“Randomized Motif Search,***

***Gibbs Sampler and Median String algorithms“***

March 14 2024

Group Members

Aimen Daddi 150119879

Mert Özincegedik 150119643

Murat Tüzün 150119633

1. **Introduction**

In this project, we implemented and compared three different algorithms commonly used for motif finding in DNA sequences: Randomized Motif Search, Gibbs Sampler, and Median String algorithms. The objective was to analyze the performance of these algorithms in terms of both execution time and the quality of the results obtained. We mutate k = "AAAAAAAAAA" string in 4 random positions and added every line as in the instruction. This is how we implemented the algorithms:

**For the Randomized Motif Search:**

1- Initialization: Randomly select an initial set of motifs (subsequences of length k) from each DNA sequence.

2- Profile Creation: Construct a profile matrix from these motifs, representing the frequency of each nucleotide at every position in the motifs.

3- Motif Selection: Use the profile to select new motifs from the DNA sequences, choosing the k-mer in each sequence that is most likely under the profile.

4- Iteration: Repeat steps 2 and 3, updating the profile with the newly selected motifs and then choosing new motifs based on this updated profile.

5- Convergence: Continue iterating until the process stabilizes (the motifs no longer change or improve significantly) or after a predefined number of iterations.

6-Outcome: The final set of motifs represents the algorithm's best guess for a common pattern across the sequences, aimed at minimizing discrepancies from a consensus motif

**For the Gibbs Sampler:**

1-Initialization: Randomly select an initial motif from each DNA sequence. This set of motifs serves as the starting point for the iterative optimization process.

2-Iteration Loop: The algorithm iteratively updates the set of motifs to find those that are most representative of common patterns across the sequences.

3-Randomly Exclude One Motif: In each iteration, randomly choose one DNA sequence and exclude its current motif from consideration. This temporarily reduces the motif set, focusing on the remaining motifs.

4-Create a Profile: Generate a profile matrix from the current set of motifs (excluding the one). This profile represents the frequency of each nucleotide at each position, adjusted with pseudocounts to avoid zero probabilities.

5-Calculate Probabilities and Select a New Motif: For the excluded sequence, calculate the probability of each possible k-mer based on the current profile. A new motif is then selected probabilistically, favoring k-mers that better match the profile.

6-Update and Evaluate: Replace the excluded motif with the newly selected one, update the motif set, and evaluate its overall quality or "score". This score reflects how well the current set of motifs matches across all sequences.

7-Track Best Motifs: Keep track of the best scoring set of motifs found so far. If a new set of motifs results in a better score, it becomes the new "best" set.

8-Stopping Criteria: The loop continues for a fixed number of iterations or stops early if it achieves a stable solution where the score no longer improves significantly.

9-Statistics and Final Output: At the end, the algorithm outputs the best motifs found, along with their score, and also provides a summary including the best score, average score, and maximum score observed during the process.

**For Median String:**

1-Read DNA Sequences: Load DNA sequences from a file. These sequences are the dataset within which we aim to find the median string.

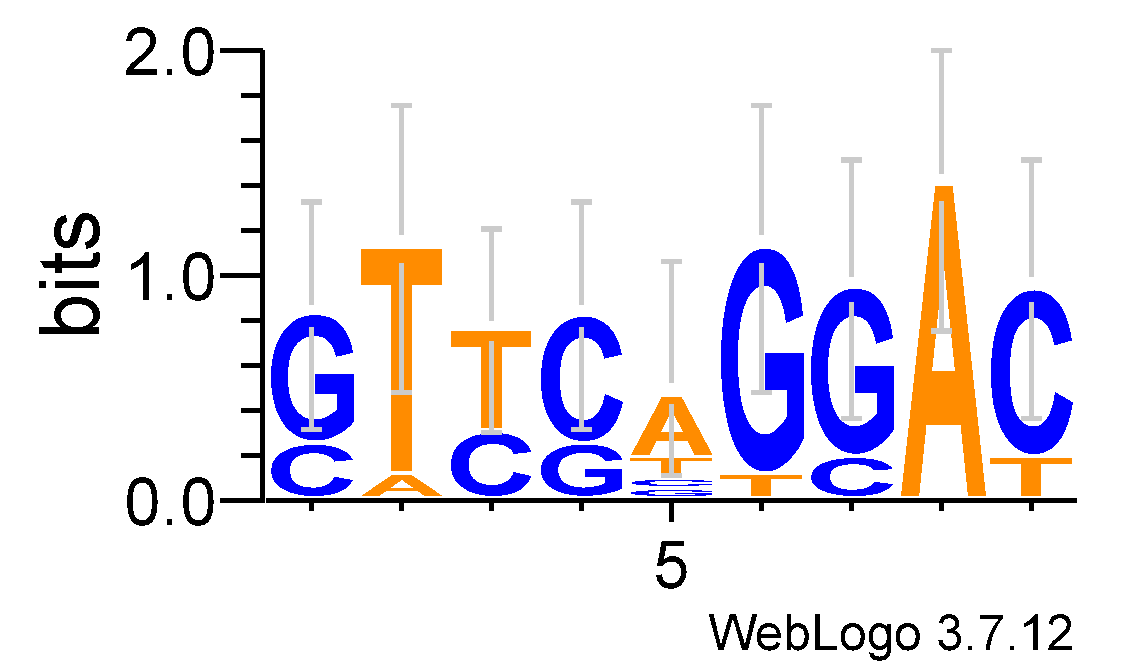
2-Hamming Distance: A function that calculates the Hamming distance between two strings of equal length, s1 and s2. The Hamming distance is the number of positions at which the corresponding symbols are different.

3-Generate All Possible k-mers: Generates all possible strings of length k made up of the nucleotides A, C, G, and T. This exhaustive list includes every combination, such as "AAA", "AAC", ..., "TTT" for k=3.

4-Find the Median String: Iterates through each possible k-mer and calculates its total Hamming distance to all DNA sequences. For each DNA sequence, it finds the minimum Hamming distance between the k-mer and any substring of length k within the sequence. The k-mer with the lowest total distance across all DNA sequences is considered the median string.

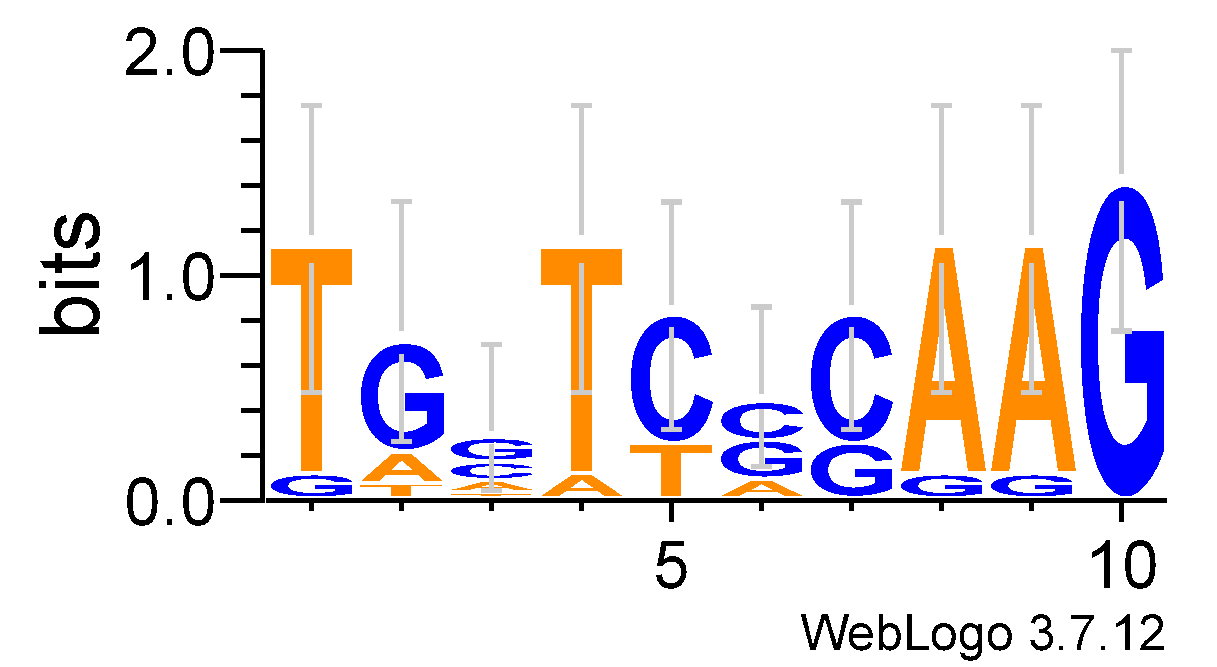
5-Execution Time Measurement: Tracks the start and end time around the median string calculation to measure the algorithm's execution time.



1. **Randomized Motif Search**
2. K=9

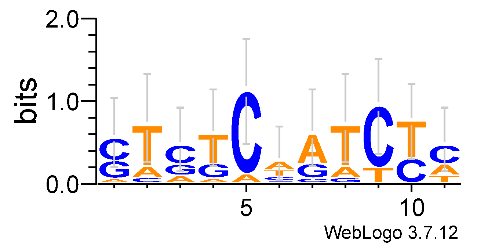
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Best Score | Max Score | Average Score | Execution Time | Consensus String |
| 34 | 53 | 34.365 | 0.420 | CCGAAACAG |
| 26 | 48 | 26.472 | 0.422 | AGTGGATAT |
| 30 | 58 | 30.704 | 0.429 | ATGGTCTAC |
| 19 | 54 | 19.907 | 0.451 | TGGGTCTAA |
| 24 | 50 | 24.509 | 0.430 | GGTCCTGCA |
| 26 | 50 | 26.709 | 0.443 | CCTCTGTTG |
| 26 | 57 | 26.596 | 0.426 | ACGGTTCAT |
| 26 | 54 | 26.630 | 0.433 | GTTCCCGCA |
| 25 | 56 | 25.717 | 0.460 | CAGTTGCCC |
| 20 | 54 | 20.982 | 0.453 | GTTCAGGAC |
| Averages for 10 runs | | | |  |
| 25.6 | 53.4 | 26.2591 | 0.4367 |  |



1. K=10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Best Score | Max Score | Average Score | Execution Time | Consensus String |
| 30 | 57 | 30.519 | 0.443 | TAGATCACAA |
| 29 | 59 | 29.577 | 0.432 | TCAAATAATT |
| 33 | 60 | 33.566 | 0.450 | AATTTAGATG |
| 26 | 56 | 27.123 | 0.483 | GTTCTGCTAC |
| 30 | 59 | 30.623 | 0.446 | TTGAGACAAA |
| 38 | 60 | 38.434 | 0.457 | GGATCCACAC |
| 25 | 57 | 25.873 | 0.463 | ATGGTGGGAG |
| 32 | 60 | 32.538 | 0.428 | CGTGGAAGCA |
| 31 | 60 | 31.604 | 0.439 | AAGAAACCAA |
| 25 | 57 | 25.947 | 0.479 | TGGTCCCAAG |
| Averages for 10 runs | | | |  |
| 29.9 | 58.5 | 30.5804 | 0.452 |  |



1. K=11

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Best Score | Max Score | Average Score | Execution Time | Consensus String |
| 34 | 60 | 34.642 | 0.494 | ACGACAAACGT |
| 33 | 62 | 33.704 | 0.497 | CGGATTAGAAC |
| 33 | 66 | 33.667 | 0.500 | TATTAGAACTC |
| 36 | 66 | 36.66 | 0.486 | ACTTGCCACTT |
| 38 | 66 | 38.566 | 0.484 | GCCTATCAGTT |
| 36 | 70 | 36.722 | 0.486 | TTGACACTCCG |
| 43 | 64 | 43.404 | 0.472 | ATACGACCCAC |
| 38 | 65 | 38.623 | 0.481 | GAGGGCCGGTC |
| 36 | 65 | 36.764 | 0.513 | CGGACGATCTT |
| 42 | 66 | 42.547 | 0.505 | CTCTCTATCTC |
| Averages for 10 runs | | | |  |
| 36.9 | 65 | 37.5299 | 0.4918 |  |

**Analysis for Randomized Motif Search:**

For execution time we noticed a consistent increase in execution time as k increases from 9 to 11, with times ranging from approximately 0.437 seconds for k=9 to about 0.492 seconds for k=11. This increase shows that the algorithm requires more computational time to process motifs as they get longer.

For the scores we noticed that as the motifs get longer the scores increase, ranging from an average of 25.6 for 10 runs for k=9 to about 36.9 for k=11. Also, we notice that the average scores are nearly the same as the best scores.

Example output for 10-mer from our code. Execution stops when score remains the same for 50 consecutive iterations. When score remains the same for some iteration( in our case its 50) it means that algorithm stuck in the local optimum.

Iteration 55:

Motifs:

G A T G C A C T G A

G A C G C C T C T A

T A A G C A C C G C

C A C G C A T C G A

T A A A C A A C G A

A A T A C A T C G T

T A A A C G A C G A

G A C A C G T C G T

A A T G C A T C T A

G A T G C A T C G T

Profile:

A: 0.21428571428571427 0.7857142857142857 0.2857142857142857 0.35714285714285715 0.07142857142857142 0.5714285714285714 0.21428571428571427 0.07142857142857142 0.07142857142857142 0.5

C: 0.14285714285714285 0.07142857142857142 0.2857142857142857 0.07142857142857142 0.7857142857142857 0.14285714285714285 0.21428571428571427 0.7142857142857143 0.07142857142857142 0.14285714285714285

G: 0.35714285714285715 0.07142857142857142 0.07142857142857142 0.5 0.07142857142857142 0.21428571428571427 0.07142857142857142 0.07142857142857142 0.6428571428571429 0.07142857142857142

T: 0.2857142857142857 0.07142857142857142 0.35714285714285715 0.07142857142857142 0.07142857142857142 0.07142857142857142 0.5 0.14285714285714285 0.21428571428571427 0.2857142857142857

Score: 30

Final Motifs:

G A T G C A C T G A

G A C G C C T C T A

T A A G C A C C G C

C A C G C A T C G A

T A A A C A A C G A

A A T A C A T C G T

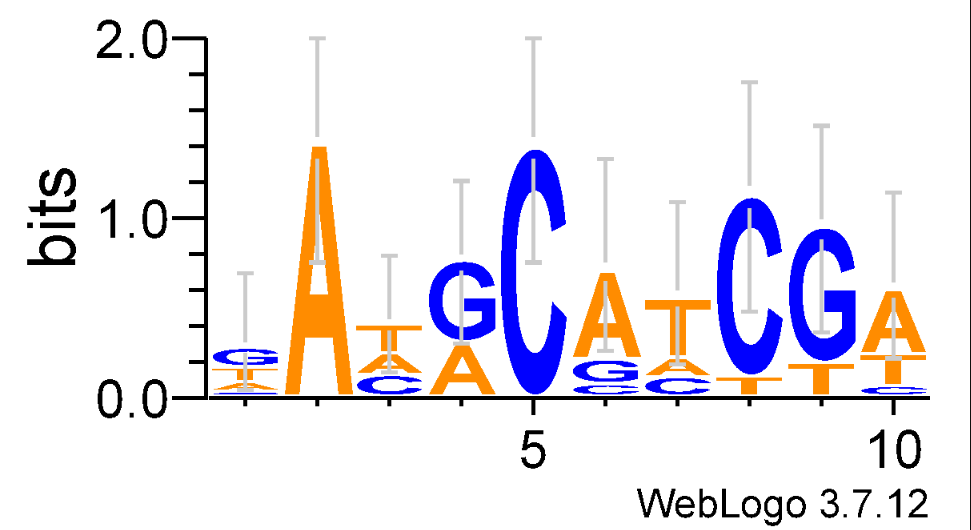
T A A A C G A C G A

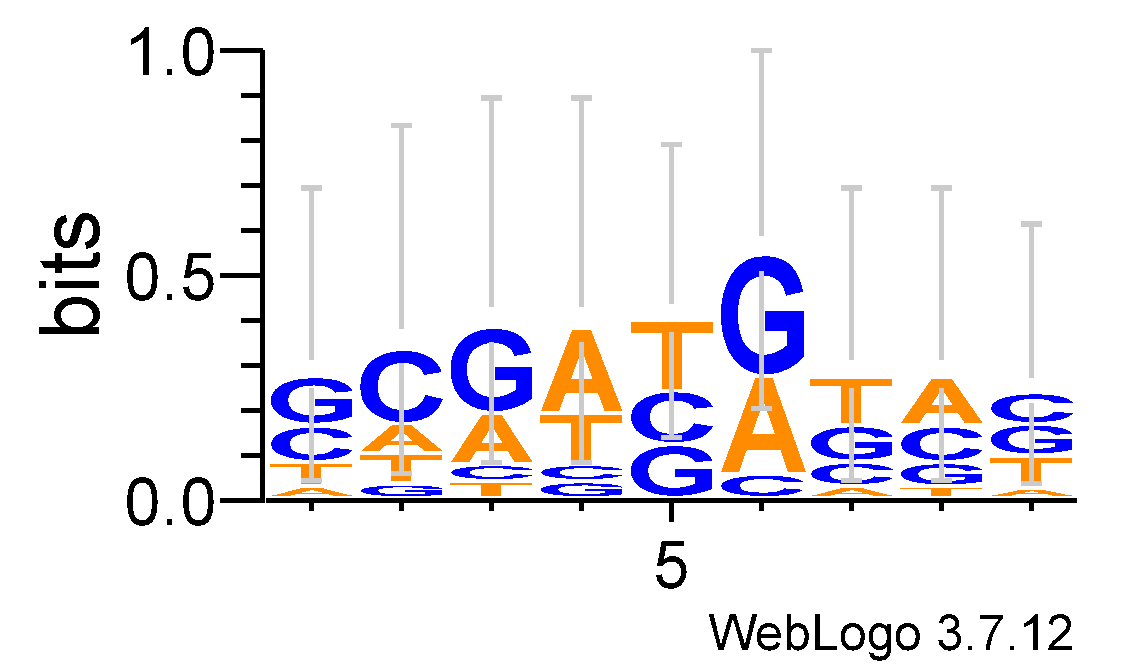
G A C A C G T C G T

A A T G C A T C T A

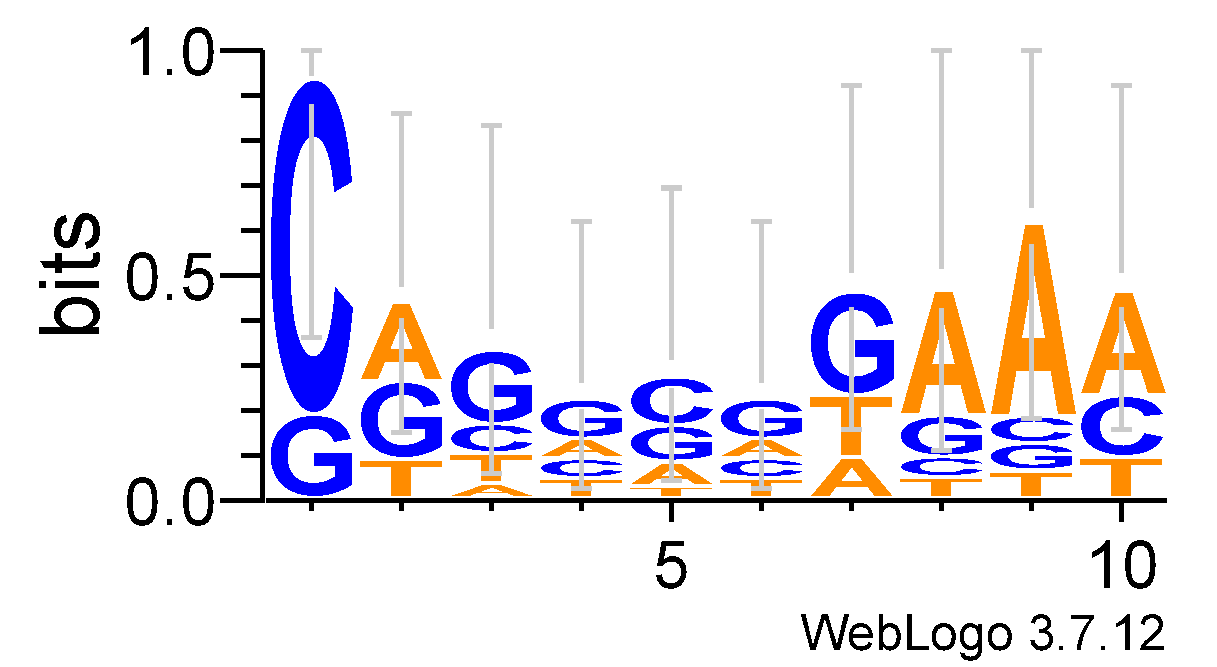
G A T G C A T C G T

Consensus String: GATGCATCGA

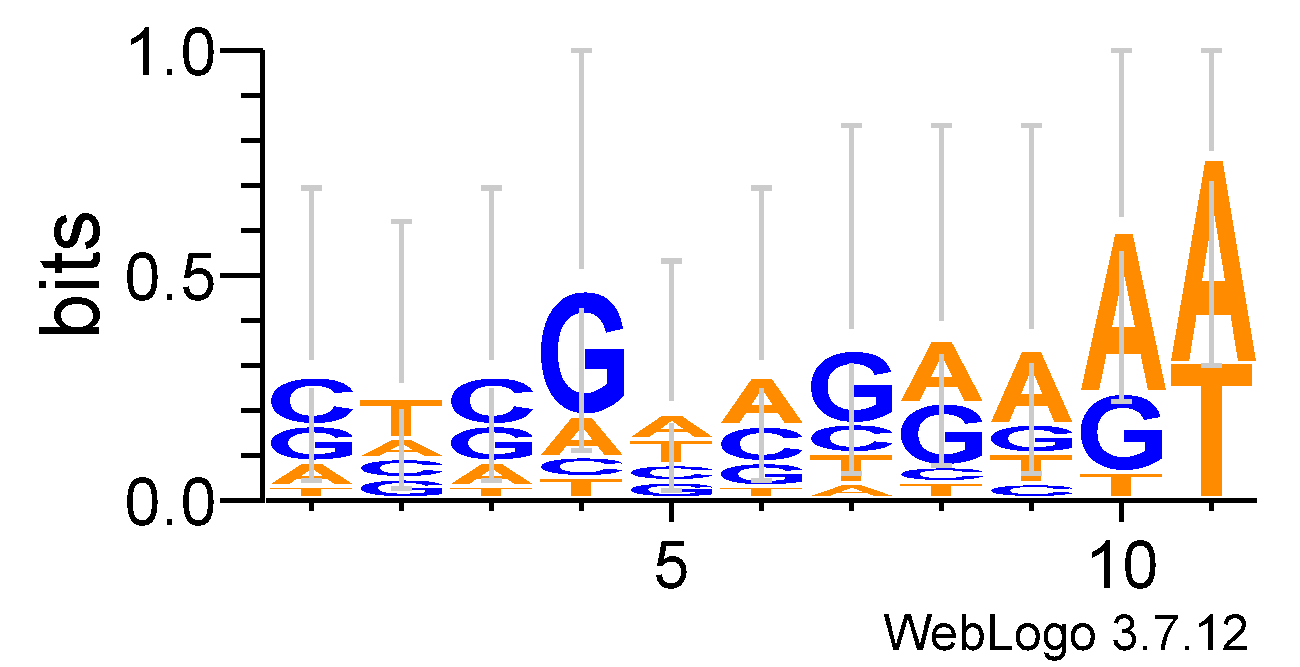


1. **Gibbs Sampler**
2. K=9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Best Score | Max Score | Average Score | Execution Time | Consensus String |
| 48 | 57 | 55.000 | 0.01496 | ACAAGATCT |
| 49 | 57 | 53.293 | 0.01496 | CAAGCAGGA |
| 51 | 58 | 56.462 | 0.01297 | GAGACGACG |
| 54 | 59 | 56.092 | 0.01795 | ATAGCGGAA |
| 54 | 58 | 55.944 | 0.01347 | AATGATCTA |
| 53 | 56 | 54.278 | 0.01396 | AACGCAAGC |
| 49 | 57 | 55.250 | 0.01297 | ACACCAGTA |
| 51 | 57 | 55.962 | 0.01296 | GGACAATGA |
| 51 | 57 | 56.091 | 0.01396 | GTGAAATAA |
| 51 | 57 | 56.392 | 0.01297 | GCGATGTAC |
| Averages for 10 runs | | | |  |
| 51.1 | 57.3 | 55.4764 | 0.014113 |  |

1. K=10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Best Score | Max Score | Average Score | Execution Time | Consensus String |
| 54 | 59 | 58.157 | 0.01320 | CCGCACAGTG |
| 56 | 62 | 60.286 | 0.01895 | GGAGAAAAAA |
| 54 | 61 | 57.617 | 0.01596 | ATAGGCGGAA |
| 54 | 64 | 59.710 | 0.01696 | CTTGACTATA |
| 54 | 61 | 59.765 | 0.01305 | GATGGATCTG |
| 53 | 61 | 57.237 | 0.01647 | GTAGAGGATA |
| 57 | 61 | 60.525 | 0.01647 | ATCGAAGCAA |
| 52 | 64 | 61.750 | 0.01296 | CTTGCGTCAC |
| 59 | 62 | 61.066 | 0.01596 | GTAGCGATAA |
| 48 | 61 | 58.271 | 0.01595 | CGGGGGGAAA |
| Averages for 10 runs | | | |  |
| 54.1 | 61.6 | 59.4384 | 0.015593 |  |

1. K=11

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Best Score | Max Score | Average Score | Execution Time | Consensus String |
| 63 | 69 | 65.937 | 0.01747 | GTAGAAAAAAT |
| 58 | 67 | 63.722 | 0.01496 | GTGGTATAATT |
| 59 | 67 | 66.392 | 0.01297 | AAAACTCTTAG |
| 60 | 67 | 65.528 | 0.01496 | CATACAATTAA |
| 60 | 68 | 66.078 | 0.01396 | CCACTAATTCA |
| 60 | 67 | 65.148 | 0.01396 | GGTAGAGTGTT |
| 60 | 67 | 64.603 | 0.01696 | GGCGGGAATAT |
| 63 | 69 | 66.387 | 0.01647 | GTAGATGAAAT |
| 65 | 72 | 67.769 | 0.01396 | CACGCCGGGAA |
| 59 | 67 | 65.016 | 0.01596 | GTCGAAGGAAA |
| Averages for 10 runs | | | |  |
| 60.7 | 68 | 65.658 | 0.015163 |  |

**Analysis for Gibbs Sampler:**

For the execution time it has significantly lower execution times for all k values compared to Randomized Motif Search, with times never exceeding 0.02 seconds. This indicates a much faster performance.

For the scores we also noticed that as the motifs get longer the scores increase, ranging from an average of 51.1 for 10 runs for k=9 to about 60.7 for k=11. Also, the scores are more consistent for the 10 runs for each k-Mer, we see the reoccurrence of the scores 51,54,60 four times each.

Example output for 10-mer from our code. Execution stops when score remains the same for 50 consecutive iterations but if the score does not remains the same for 50 consecutive iterations, it will execute 1000 times. It is almost impossible for Gibbs Samplers to have the same score for 50 consecutive iterations but we want to show that Gibbs Sampler will give different results and it will never stuck.  
  
Iteration 1000: Best Score: 52, Average Score: 60.87312687312687

Profile Matrix:

A: 0.39 0.28 0.28 0.28 0.28 0.28 0.39 0.50 0.39 0.61

C: 0.39 0.17 0.39 0.06 0.39 0.28 0.28 0.17 0.28 0.06

G: 0.28 0.39 0.28 0.72 0.39 0.39 0.39 0.39 0.28 0.28

T: 0.17 0.39 0.28 0.17 0.17 0.28 0.17 0.17 0.28 0.28

Scores Summary: Best Score: 52, Max Score: 61, Average Score: 60.87312687312687

Motifs:

AGTGCATGTA

AGATAATGGG

GGTGTTTAAC

TGATCGAGCA

CATTATTTAA

CATGCCTGAA

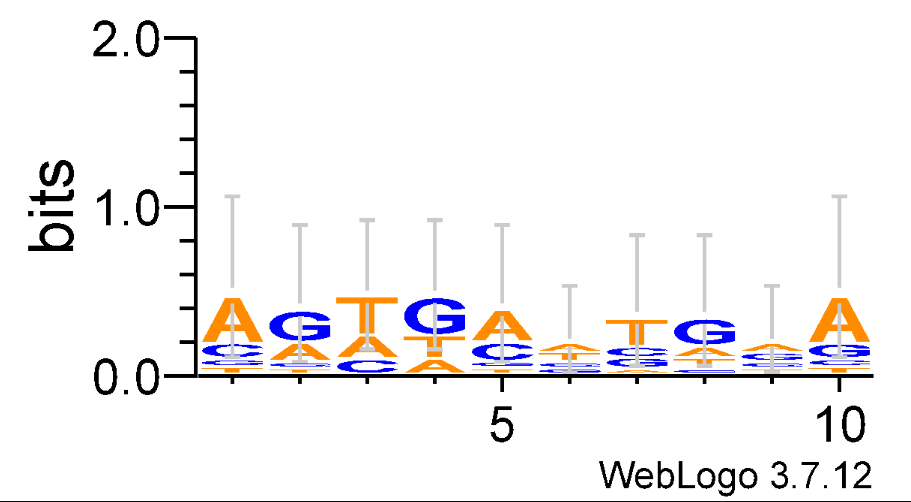
ATTGATCTCT

AAAGAGGATA

ACCAACGGGA

AGCAGACCGG

Consensus String: AGTGAATGGA



1. **Median String**
2. K=9



1. K=10

****

1. K=11



1. Executions time from 1 to 11 Mers:

|  |  |
| --- | --- |
| K-mers | Execution Time(s) |
| 1 | 0.0109 |
| 2 | 0.0498 |
| 3 | 0.2188 |
| 4 | 0.9716 |
| 5 | 4.0509 |
| 6 | 17.308 |
| 7 | 73.978 |
| 8 | 315.97 |
| 9 | 1342.5 |
| 10 | 7564.5 |

Analysis for the median string:

We notice that it takes the longest time by far for executing, also as the motifs get longer the time increases drastically. We ran it from 1 to 11 k-Mer so we understand how It increases, by looking at the graph, we see that the execution time increases exponentially, which suggest extremely high computational demands for larger k values.

1. **Conclusion**

Execution Time: The Gibbs Sampler is the fastest, offering fast performance for all k values. Randomized Motif Search shows reasonable times but gets slower ask increases. Median String potentially becomes impractical for large k due to exponential time increases.

Consensus Strings: Both Randomized Motif Search and Gibbs Sampler produce meaningful consensus strings that evolve with k, suggesting their utility in motif discovery across varying lengths.