



**MARMARA**  
**UNIVERSITY**

# **CSE4065 - Introduction to Computational Genomics**

Assignment 1 Report  
November 18, 2021

Yusuf Talha Erdem (150117012)  
Sena Dilara Yangöz (150119706)  
Ali Reza Ibrahimzada (150119870)

Prior to executing the algorithms on the randomly generated DNA strings, we randomly generated the motifs, DNA strings, positions of the mutations, base pair mutations, positions of the mutated motifs in the DNA string, etc. Then for the rest of the project, we simply use the same DNA strings we have generated in the beginning of the project.

## Randomized Motif Search

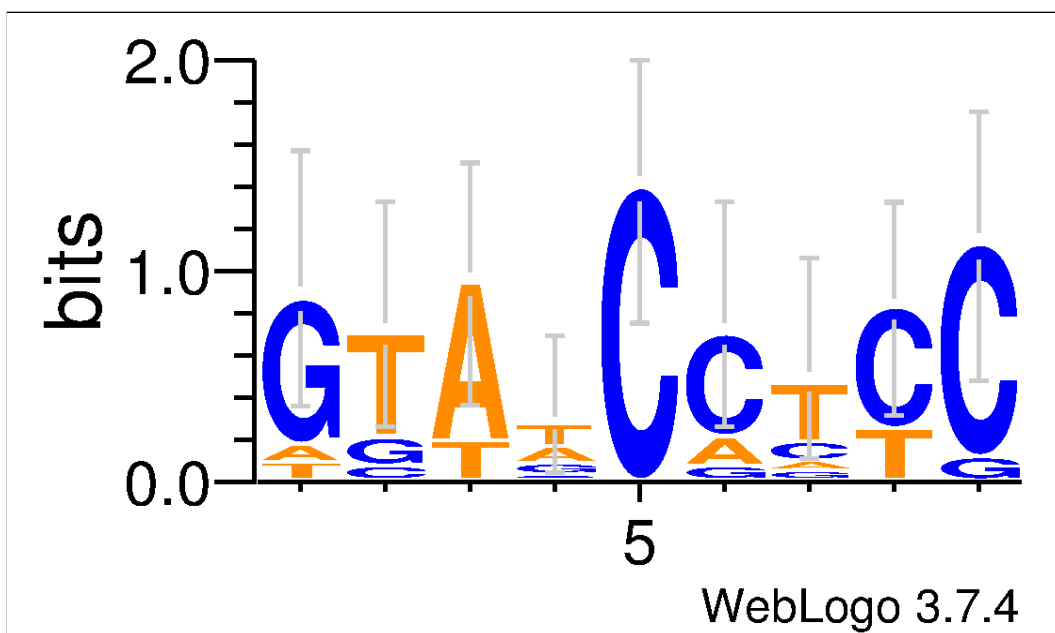
For the Randomized Motif Search algorithm, we started by choosing 10 random motifs from the DNA strings. Then, we randomly select initial motifs from the DNA strings, set it to best motifs, and calculate a profile based on the initial motifs using the *get\_profile* function. Furthermore, we call the *get\_likely\_motifs* function in order to extract the set of kmers with the highest probability to be our motifs. The function runs a sliding window over the DNA string and calculates the probability at each iteration. Later at the end, it returns the set of most probable motifs from the DNA strings. At the end, we check if there is any improvement in the overall score of the algorithm compared to the previous iteration. That is, if there is any improvement in the score, we re-iterate and use the new motifs as our best one, otherwise we end the iteration and return the current best scores. What we have explained so far describes a single run of the Randomized Motif Search algorithm, we continue doing this for a total of 10 runs and report different statistics about this iterative algorithm at the end.

**K = 9:**

**Original Motif (Randomly Generated):** ATGGAGTCTA

**Algorithm Output (Consensus String):** GTATCCTCC-

```
ali@ali: ~  
ali@ali:~$ python3 main.py -t 10 -l 10 -d 4 -n 500 -k 9 -total_iter 10 -input_f  
ile dna_strings.txt -generate_data False -algorithm randomized  
original motif: atggagtcta  
max score: 30  
average score: 26.8  
best score: 24  
score: 24  
best motifs:  
ggagcctcc  
gtttccctc  
gtatcatcc  
atatacctcg  
gtacccacc  
gtaacgtcc  
gcatccttc  
gttaccgtc  
ttagcatcc  
ggaaccccc  
average running time per iteration (s): 0.15  
total running time (s): 1.54  
ali@ali:~$
```

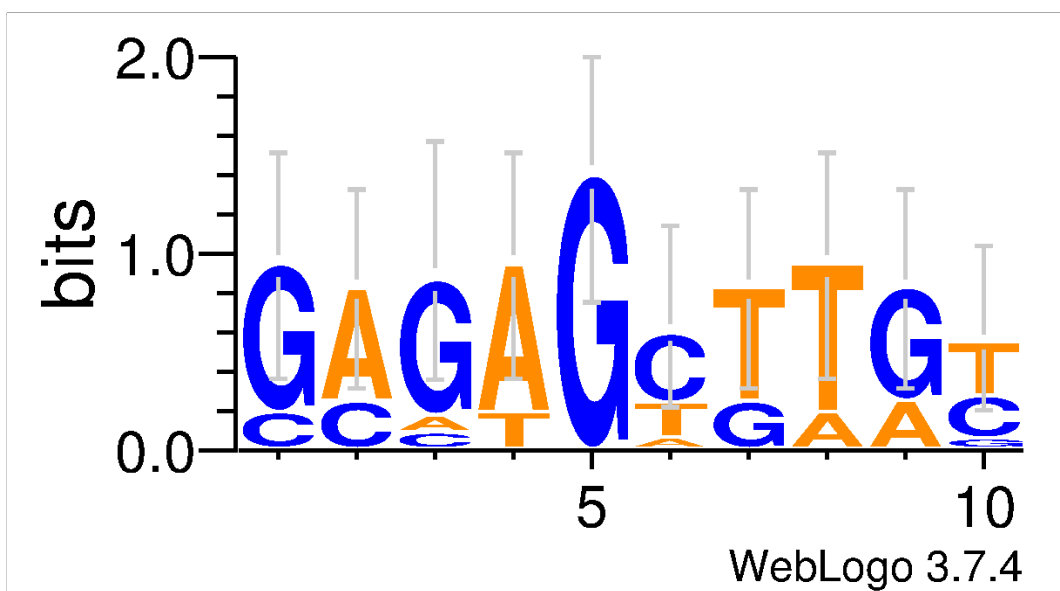


K = 10:

**Original Motif (Randomly Generated):** ATGGAGTCTA

**Algorithm Output (Consensus String):** GAGAGCTTGT

```
ali@ali: ~  
ali@ali:~$ python3 main.py -t 10 -l 10 -d 4 -n 500 -k 10 -total_iter 10 -input_  
file dna_strings.txt -generate_data False -algorithm randomized  
original motif: atggagtcta  
max score: 37  
average score: 32.6  
best score: 25  
score: 25  
best motifs:  
gagagttagt  
gagagtttac  
cagagcgagt  
gcgagagtgg  
gaaagtttgc  
gagtgcctgc  
cagagcttat  
gagtgcctgt  
gcgagcgtac  
gccagcttgt  
average running time per iteration (s): 0.14  
total running time (s): 1.41  
ali@ali:~$
```

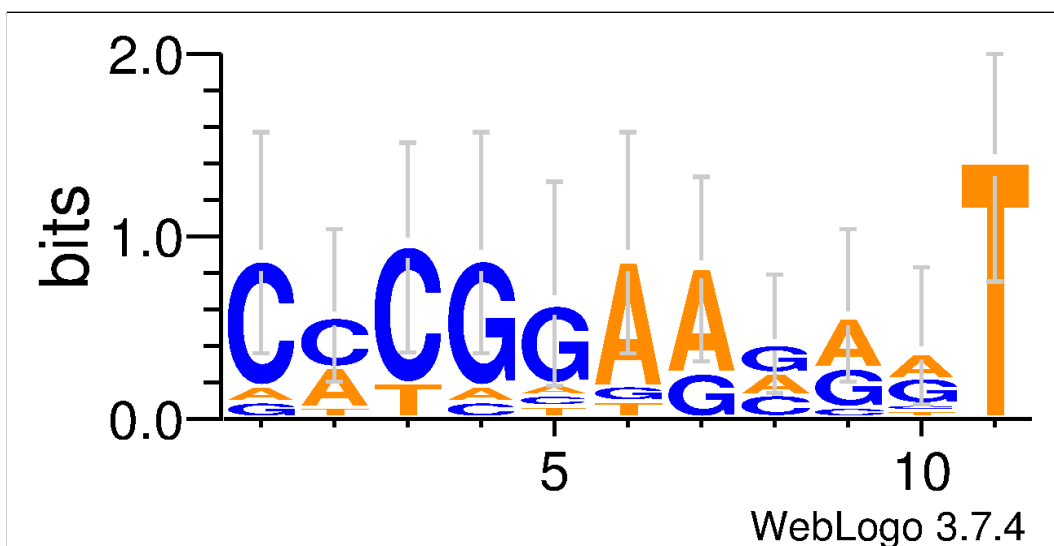


**K = 11:**

**Original Motif (Randomly Generated):** ATGGAGTCTA-

**Algorithm Output (Consensus String):** CCCGGAAGAAT

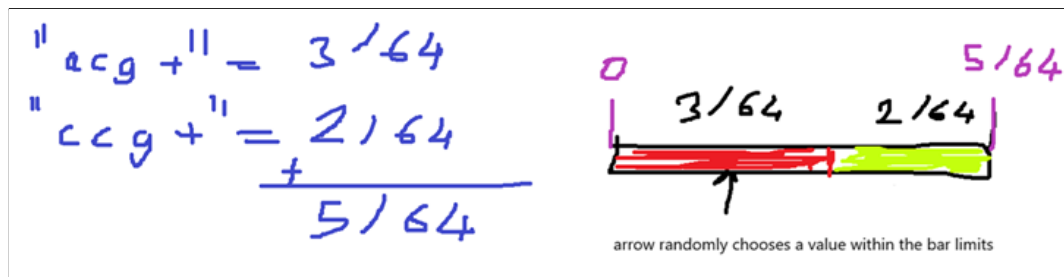
```
ali@ali: ~  
ali@ali:~$ python3 main.py -t 10 -l 10 -d 4 -n 500 -k 11 -total_iter 10 -input_  
file dna_strings.txt -generate_data False -algorithm randomized  
original motif: atggagtcta  
max score: 42  
average score: 39.4  
best score: 36  
score: 36  
best motifs:  
cccggagagtt  
cctgtaagcat  
cctggaacagt  
cccgcgagagt  
cacgaaaagat  
gacagaacgat  
acccgtagagt  
cacggagcagt  
cacggagaaat  
ctcggaaggct  
average running time per iteration (s): 0.16  
total running time (s): 1.62  
ali@ali:~$
```



## Gibbs Sampling

For the Gibbs Sampling algorithm, we started by choosing 10 random motifs from the DNA strings like we did in Randomized Motif Search. Then, since one of those motifs should be eliminated randomly, we randomly selected the motif to be eliminated and removed it from the list which contains the initial motifs. After creating a profile with the *get\_profile* function for the rest of the 9 motifs, with the *calculate\_prob\_of\_deleted\_string* function the probabilities of the k-mers which are in the eliminated DNA strings is calculated and kept in a dictionary (key = k-mer, value= probability of the k-mer). After that, with the *roll\_dice* function a k-mer is selected randomly but the one with the highest probability is

more likely to be chosen. *roll\_dice* takes probabilities as an argument. What is stored in the probabilities is possible motives and their matches with profile. Such as "atgt" : 3/64, "ccgt" : 2/64. To randomly choose a motif, we calculate the sum of probability of all motifs. Then we choose a random value between 0 and the sum of motives and next we need to check which motif this random value is corresponding to then we return to the motif. Then the new k-mer is added to the motifs list and the score is compared with the best score. If the new score is better than the best score, the best score is synchronized to the new score and the loop counter is reset, otherwise, the loop counter is incremented by one. When the loop counter is equal to 50, this means that the score has not been improved for the last 50 iterations, so the loop ends. An illustration of how the dice rolling works is given below:

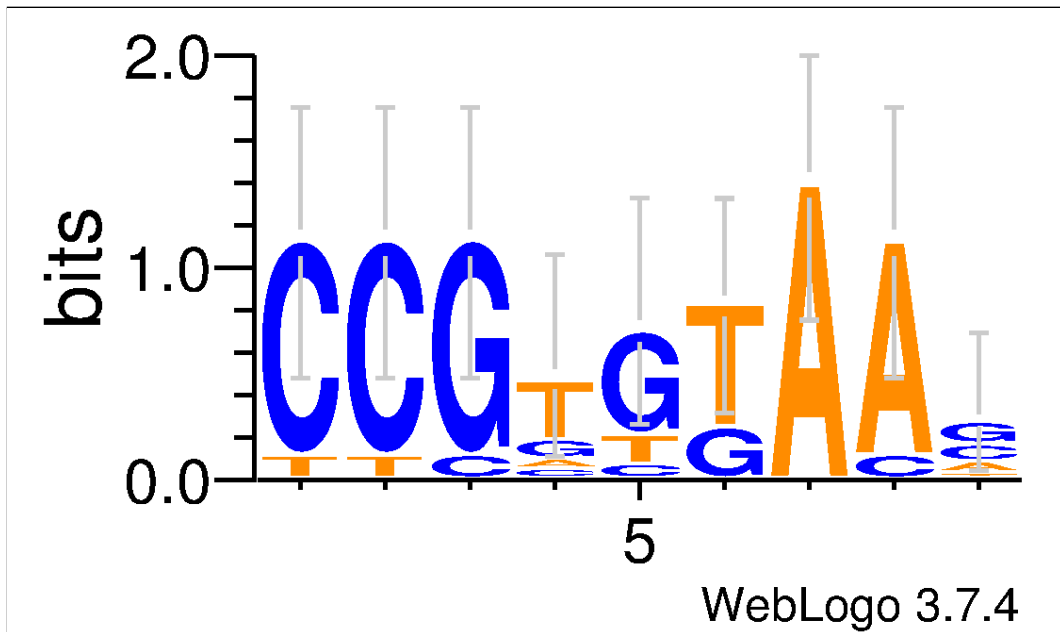


**K = 9:**

**Original Motif (Randomly Generated):** ATGGAGTCTA

**Algorithm Output (Consensus String):** CCGTGTAAG-

```
ali@ali: ~  
ali@ali:~$ python3 main.py -t 10 -l 10 -d 4 -n 500 -k 9 -gibbs_iters 50 -total_  
iter 10 -input_file dna_strings.txt -generate_data False -algorithm gibbs  
original motif: atggagtcta  
max score: 38  
average score: 29.6  
best score: 20  
score: 20  
best motifs:  
ccgcttact  
ctgaggaag  
ccgtgtaag  
tcgtgtaag  
ccgtgtaac  
ccgtctaaa  
ccggggaag  
ccgtttaac  
cccgggaac  
ccgtgtaaa  
average running time per iteration (s): 0.43  
total running time (s): 4.30  
ali@ali:~$
```

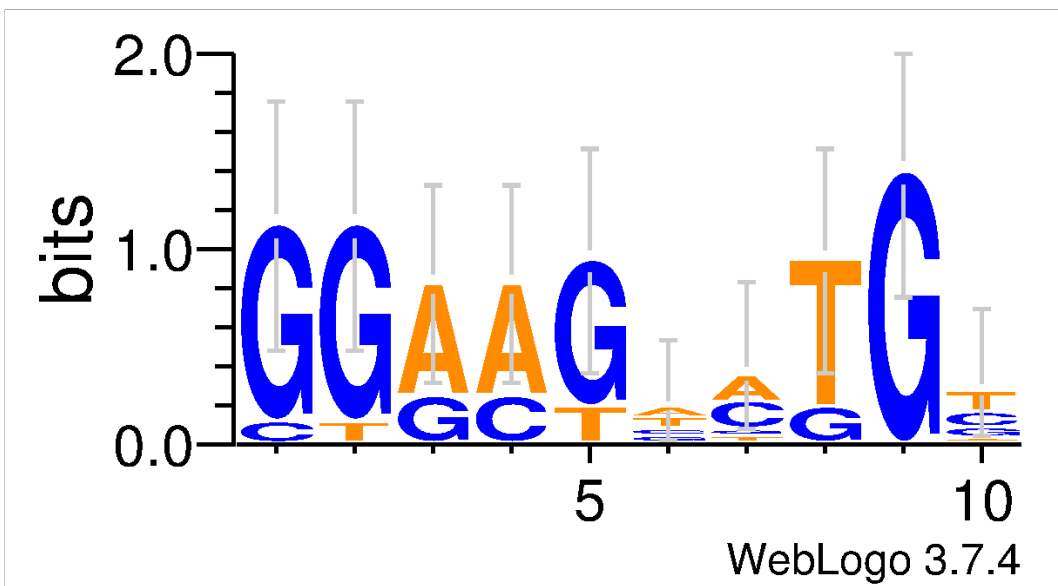


**K = 10:**

**Original Motif (Randomly Generated):** ATGGAGTCTA

**Algorithm Output (Consensus String):** GGAAGAATGT

```
ali@ali: ~  
ali@ali:~$ python3 main.py -t 10 -l 10 -d 4 -n 500 -k 10 -gibbs_iters 50 -total  
_iter 10 -input_file dna_strings.txt -generate_data False -algorithm gibbs  
original motif: atggagtcta  
max score: 44  
average score: 36.0  
best score: 31  
score: 31  
best motifs:  
ggaagtttgg  
cgacgaatgc  
ggaatgctgc  
gtaagtatgt  
gggctcatgg  
ggacgtctgt  
ggaagacgga  
gggagagtgc  
gggagcaggt  
ggaaggctgt  
average running time per iteration (s): 0.51  
total running time (s): 5.09  
ali@ali:~$
```



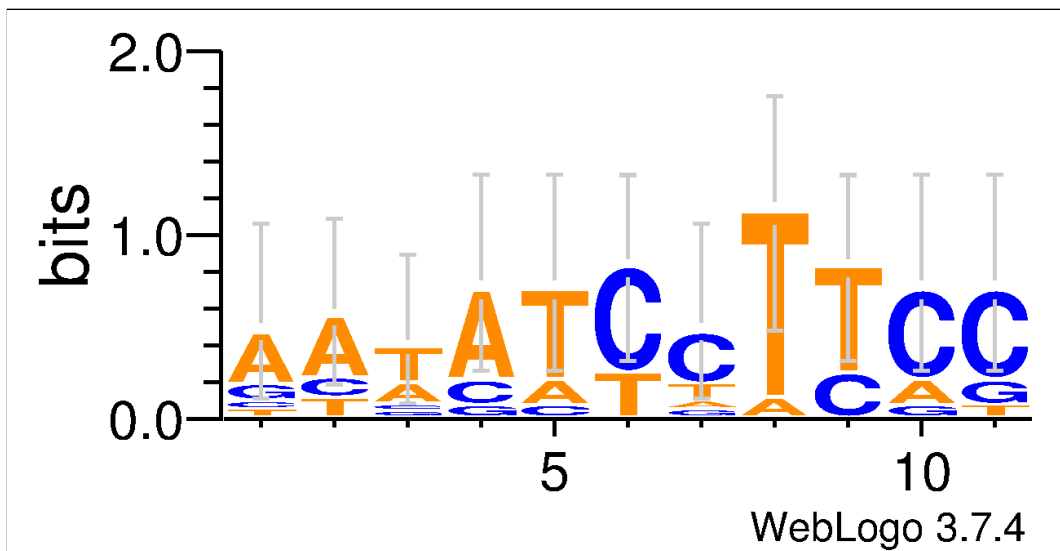
**K = 11:**

**Original Motif (Randomly Generated):** ATGGAGTCTA-

**Algorithm Output (Consensus String):** AATATCCTTCC



```
ali@ali: ~  
ali@ali:~$ python3 main.py -t 10 -l 10 -d 4 -n 500 -k 11 -gibbs_iters 50 -total  
_iter 10 -input_file dna_strings.txt -generate_data False -algorithm gibbs  
original motif: atggagtcta  
max score: 45  
average score: 40.1  
best score: 36  
score: 36  
best motifs:  
aatgattttcc  
ttcaactttcg  
attccccttcc  
catatcctcgt  
gctatcgttcc  
gaaatcaatac  
aagcttctccc  
aaaatcctcac  
aaaattcttcg  
actatccttcc  
average running time per iteration (s): 0.54  
total running time (s): 5.43  
ali@ali:~$
```



## Comparison and Conclusion

In this section, we will compare the performance of the Randomized Motif Search (RMS) with the Gibbs Sampling algorithm. In terms of the running time, Gibbs Sampling takes a significantly longer time compared to RMS because of the improvement waiting in Gibbs. In contrast, RMS does not wait for another iteration if there is no improvement in the score value. Moreover, it is visible from the score graph that as the value of  $k$  increases, the overall score of both algorithms increase as well. It can be concluded that although Gibbs takes a lot of time to produce its best motifs, RMS produces almost the same scores in a much shorter amount of time. On the other hand, we must also mention that

Gibbs algorithm has the potential to produce better scores if we let it run for longer iterations.

