CSE 4065 - Introduction to Computational Genomics
Project 2 - Report

150114051 - Mehmet MUM
150115067 - Tayfun YURDAER

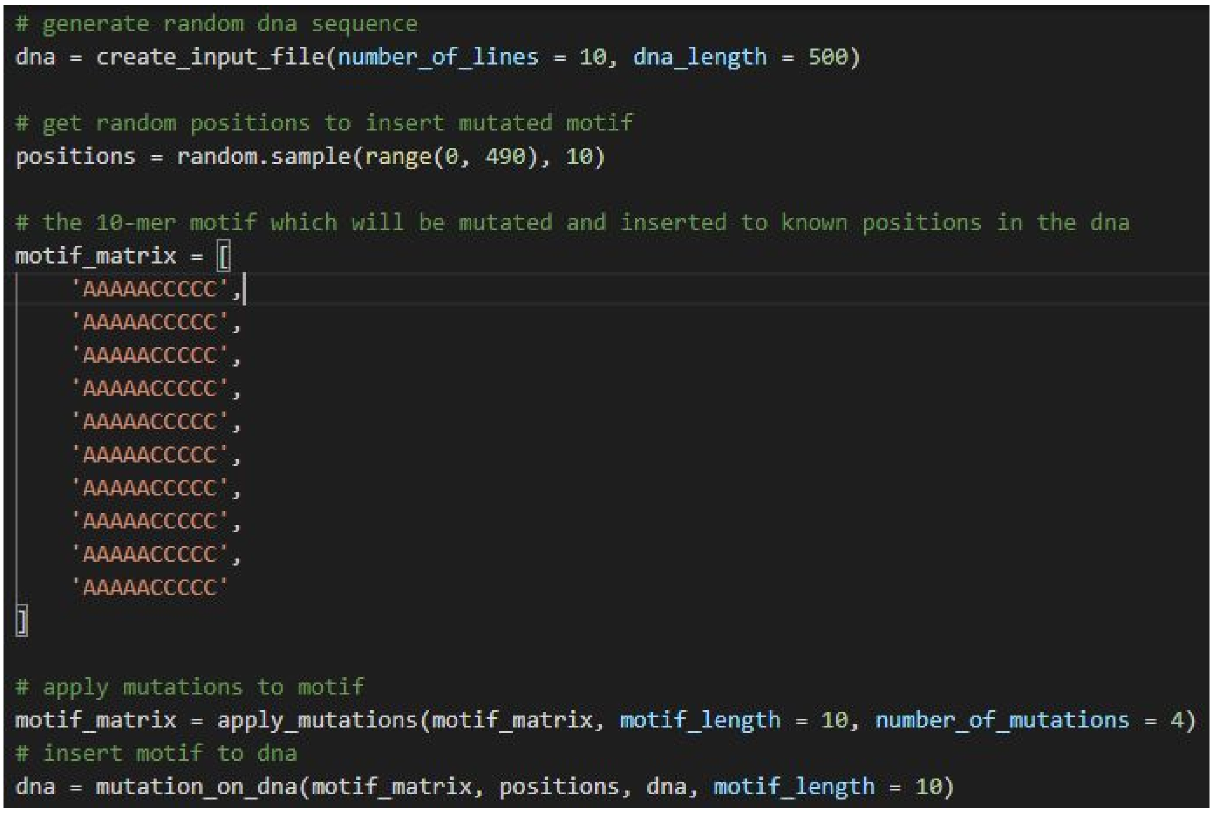
1-) Input File

First, we created dna sequence which has 10 lines. Each line consist of 500 length.

Then, we determine a 10-mer motif to insert dna. The 10-mer motif is ‘AAAAACCCCC’.

We have 10 lines, so we generate 4 random positions to apply mutation on motif. In order to get variety, for each line we generated new random positions.

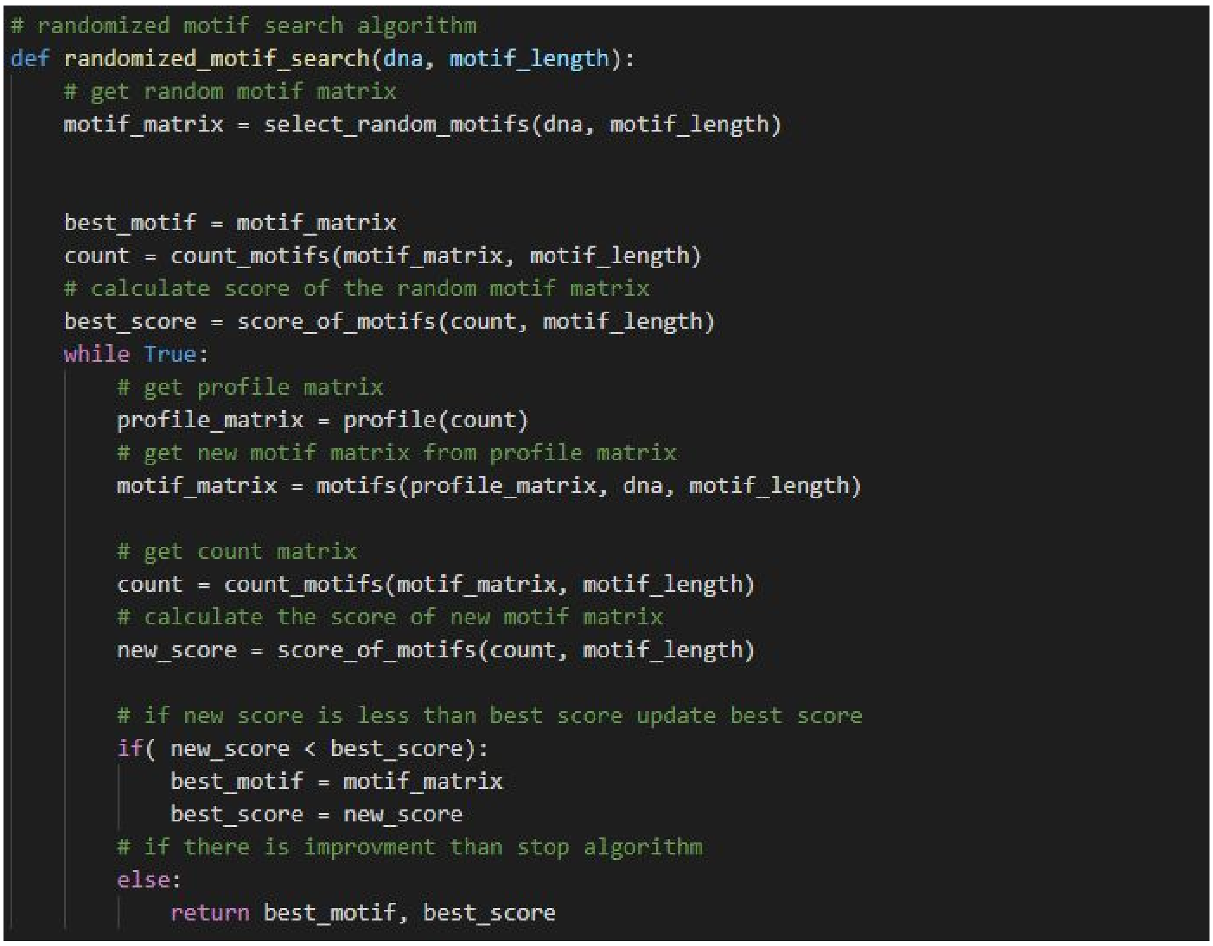
After applying the mutations, we inserted new motifs to dna.



```
# generate random dna sequence
dna = create_input_file(number_of_lines = 10, dna_length = 500)

# get random positions to insert mutated motif
positions = random.sample(range(0, 490), 10)

# the 10-mer motif which will be mutated and inserted to known positions in the dna
motif_matrix = [
    'AAAAACCCCC',
    'AAAAACCCCC',
    'AAAAACCCCC',
    'AAAAACCCCC',
    'AAAAACCCCC',
    'AAAAACCCCC',
    'AAAAACCCCC',
    'AAAAACCCCC',
    'AAAAACCCCC',
    'AAAAACCCCC'
]

# apply mutations to motif
motif_matrix = apply_mutations(motif_matrix, motif_length = 10, number_of_mutations = 4)
# insert motif to dna
dna = mutation_on_dna(motif_matrix, positions, dna, motif_length = 10)
```

2-) Randomized Motif Search - Algorithm

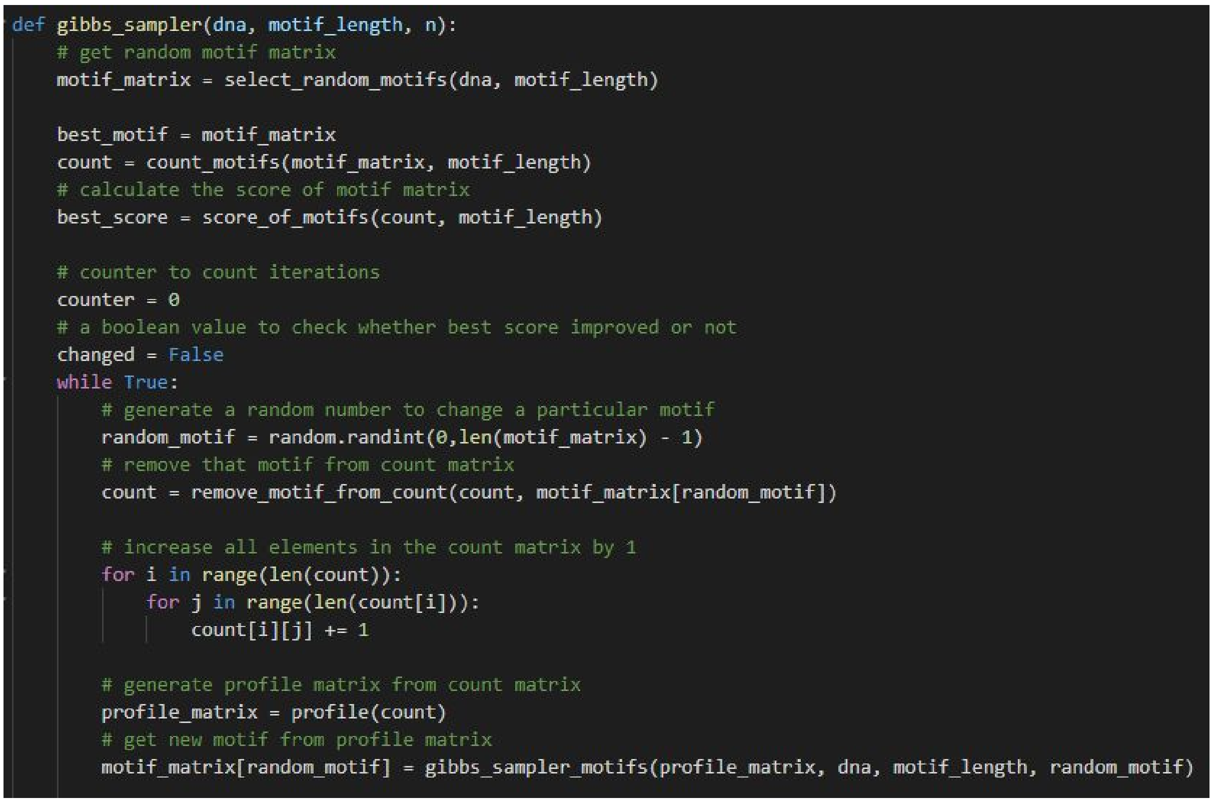
In this algorithm, first we selected random motifs from each line. Initially, best score and best motif are the random selected motifs. And then in a while loop we calculated profile matrix of this motifs, then we calculated new motif matrix from the profile matrix. For each line we take the motif which has highest probability. We calculated the score of new motifs, if the score of new motifs is less than the best score, we updated best score and best motif to new motifs and new score, then we continue the algorithm. If the score of new motifs is higher than or equal to best score, we finished the algorithm.



```python
# randomized motif search algorithm
def randomized_motif_search(dna, motif_length):
    # get random motif matrix
    motif_matrix = select_random_motifs(dna, motif_length)


    best_motif = motif_matrix
    count = count_motifs(motif_matrix, motif_length)
    # calculate score of the random motif matrix
    best_score = score_of_motifs(count, motif_length)
    while True:
        # get profile matrix
        profile_matrix = profile(count)
        # get new motif matrix from profile matrix
        motif_matrix = motifs(profile_matrix, dna, motif_length)

        # get count matrix
        count = count_motifs(motif_matrix, motif_length)
        # calculate the score of new motif matrix
        new_score = score_of_motifs(count, motif_length)

        # if new score is less than best score update best score
        if( new_score < best_score):
            best_motif = motif_matrix
            best_score = new_score
        # if there is improvment than stop algorithm
        else:
            return best_motif, best_score
```

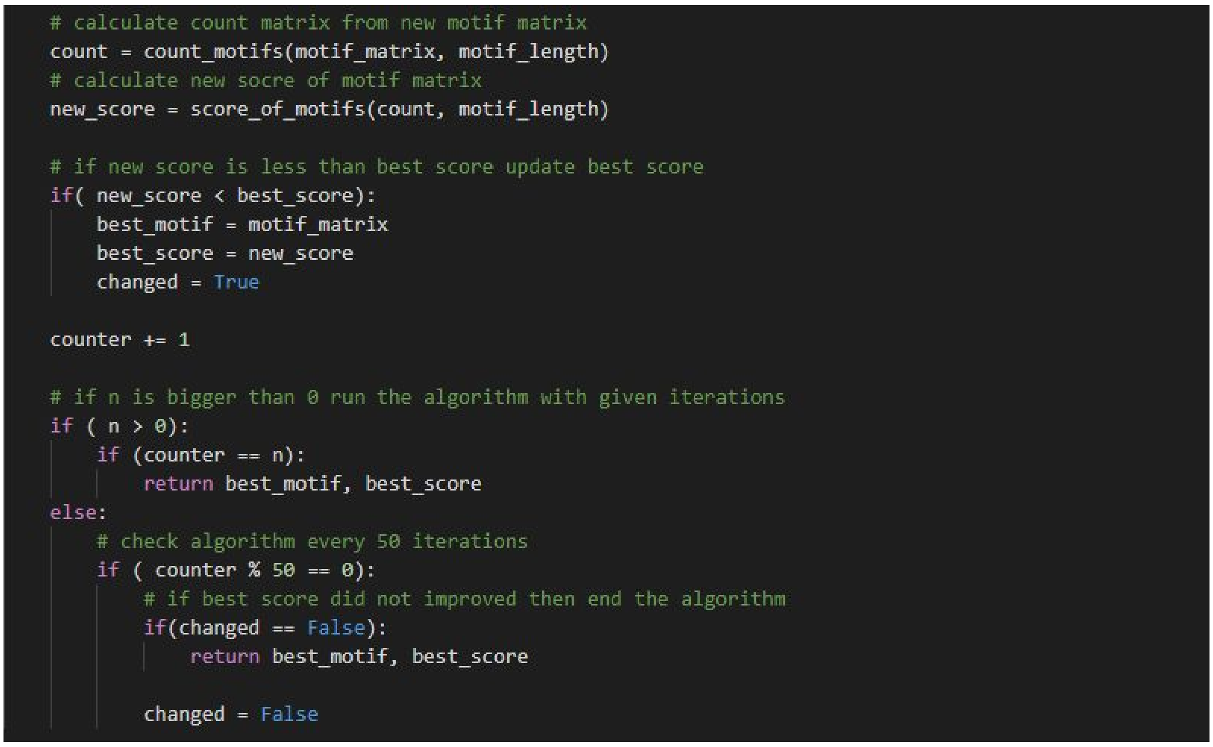
3-) Gibbs Sampler - Algorithm

In this algorithm, we again randomly selected motifs from each line. Initially, best motif and best score is this motifs. In this algorithm we have a counter to count number of iterations. In this algorithm we did not changed whole motifs as in the randomized motif search algorithm, instead we only change a single motif for each iteration. In a while loop, each iteration we randomly select a motif, we removed this motif from count matrix. And then we increased each element of count matrix by 1. We calculate profile matrix from this count matrix. Then, we calculated calculated each motif’s probability according to the profile matrix.



```python
def gibbs_sampler(dna, motif_length, n):
    # get random motif matrix
    motif_matrix = select_random_motifs(dna, motif_length)

    best_motif = motif_matrix
    count = count_motifs(motif_matrix, motif_length)
    # calculate the score of motif matrix
    best_score = score_of_motifs(count, motif_length)

    # counter to count iterations
    counter = 0
    # a boolean value to check whether best score improved or not
    changed = False
    while True:
        # generate a random number to change a particular motif
        random_motif = random.randint(0,len(motif_matrix) - 1)
        # remove that motif from count matrix
        count = remove_motif_from_count(count, motif_matrix[random_motif])

        # increase all elements in the count matrix by 1
        for i in range(len(count)):
            for j in range(len(count[i])):
                count[i][j] += 1

        # generate profile matrix from count matrix
        profile_matrix = profile(count)
        # get new motif from profile matrix
        motif_matrix[random_motif] = gibbs_sampler_motifs(profile_matrix, dna, motif_length, random_motif)
```

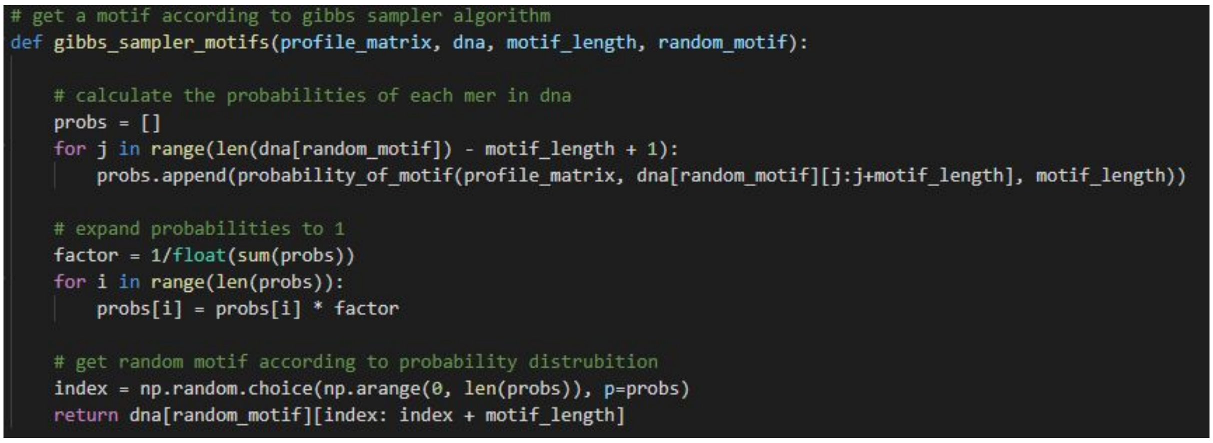
We again calculate the score of new motifs and do the same operations as in the randomized motif search. There are two types of ending condition of this algorithm. First one is, algorithm checks the best score every 50 iterations. If the best score is not improved, the algorithm ends. Second one, the algorithm takes n value and run this algorithm n times.



```
    # calculate count matrix from new motif matrix
    count = count_motifs(motif_matrix, motif_length)
    # calculate new socre of motif matrix
    new_score = score_of_motifs(count, motif_length)

    # if new score is less than best score update best score
    if( new_score < best_score):
        best_motif = motif_matrix
        best_score = new_score
        changed = True

    counter += 1

    # if n is bigger than 0 run the algorithm with given iterations
    if ( n > 0):
        if (counter == n):
            return best_motif, best_score
    else:
        # check algorithm every 50 iterations
        if ( counter % 50 == 0):
            # if best score did not improved then end the algorithm
            if(changed == False):
                return best_motif, best_score

            changed = False
```

In the randomized motif search, we select a motif which has highest probability. But, in this algorithm, we are selecting a motif randomly according to probability distribution.

In order to run random number generator with probability distribution, we expand probability values to 1.



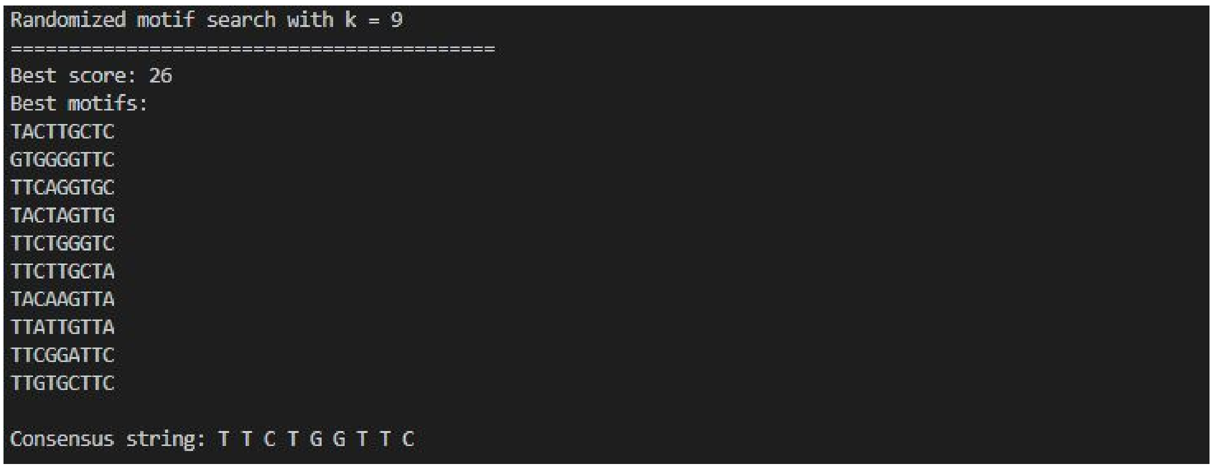
```
# get a motif according to gibbs sampler algorithm
def gibbs_sampler_motifs(profile_matrix, dna, motif_length, random_motif):

    # calculate the probabilities of each mer in dna
    probs = []
    for j in range(len(dna[random_motif]) - motif_length + 1):
        probs.append(probability_of_motif(profile_matrix, dna[random_motif][j:j+motif_length], motif_length))

    # expand probabilities to 1
    factor = 1/float(sum(probs))
    for i in range(len(probs)):
        probs[i] = probs[i] * factor

    # get random motif according to probability distrubition
    index = np.random.choice(np.arange(0, len(probs)), p=probs)
    return dna[random_motif][index: index + motif_length]
```

4-) Outputs

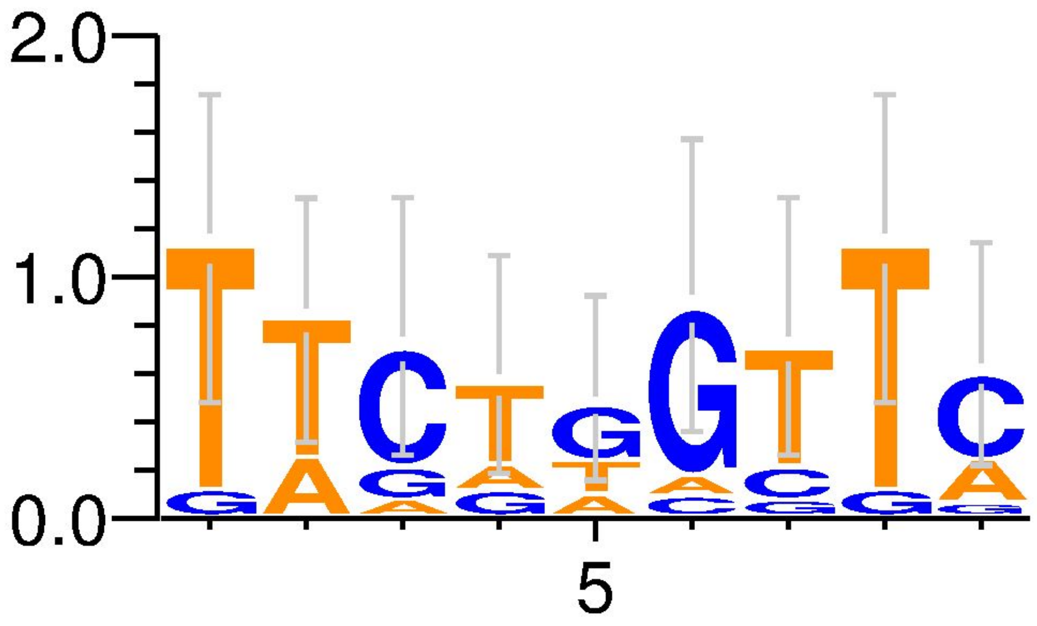
We have run the algorithms which are randomized motif search, gibbs sampler which checks every 50 iterations and gibbs sampler which runs 100000 iterations. We have run these algorithms with motif lengths 9, 10 and 11.

4.1) Run algorithms with motif length 9

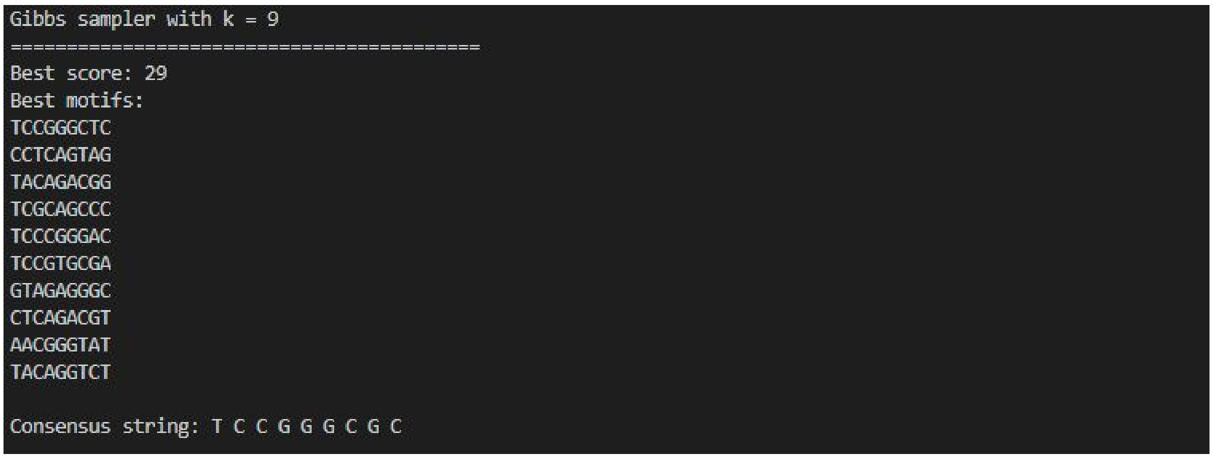
4.1.1) Randomized motif search



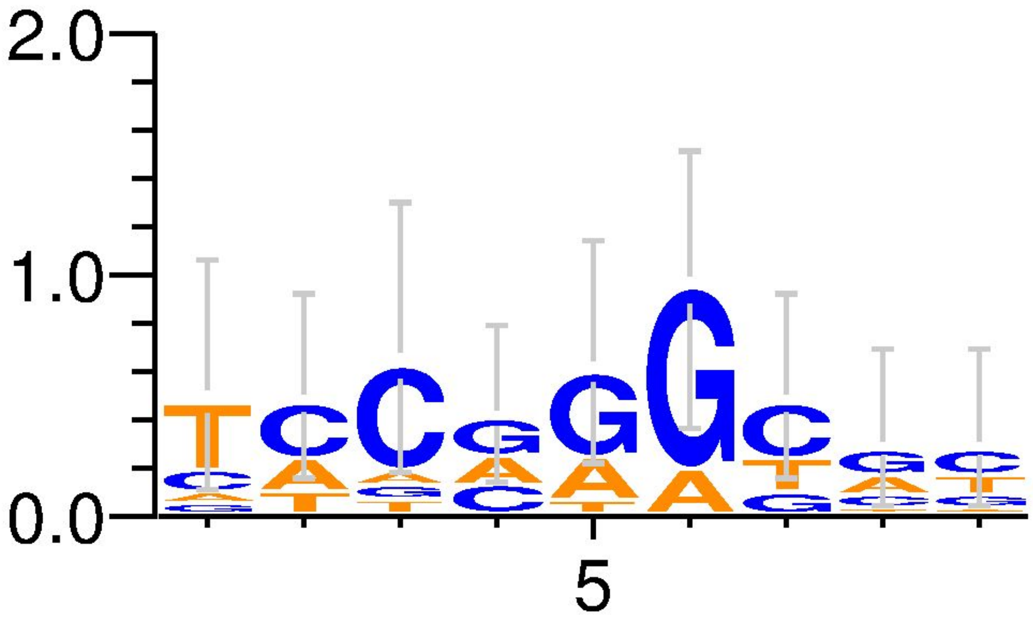
```
Randomized motif search with k = 9
==========================================
Best score: 26
Best motifs:
TACTTGCTC
GTGGGGTTC
TTCAGGTGC
TACTAGTTG
TTCTGGGTC
TTCTTGCTA
TACAAGTTA
TTATTGTTA
TTCGGATTC
TTGTGCTTC

Consensus string: T T C T G G T T C
```



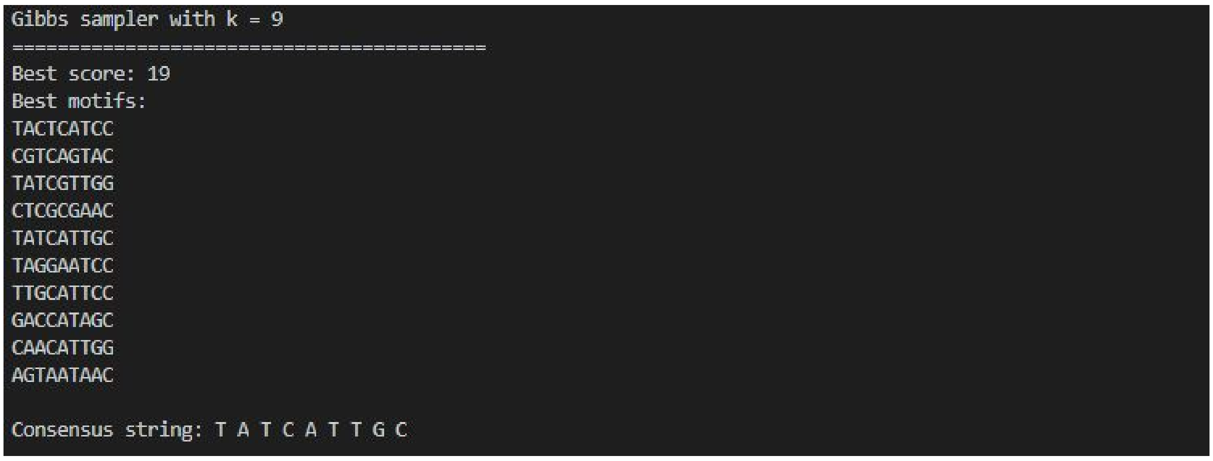
4.1.2) Gibbs sampler which check every 50 iterations



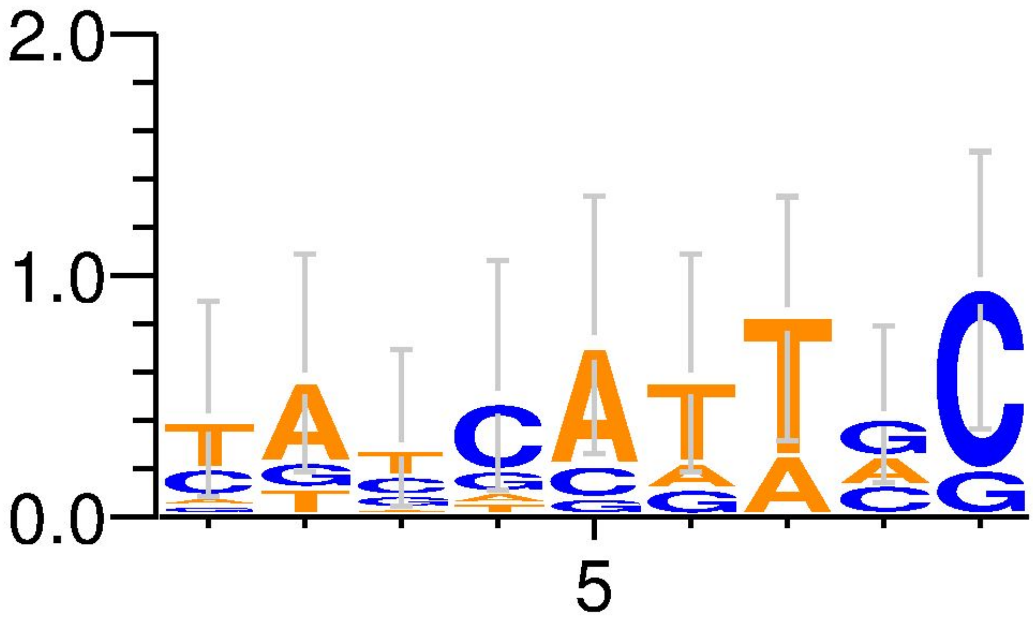
```
Gibbs sampler with k = 9
==========================================
Best score: 29
Best motifs:
TCCGGGCTC
CCTCAGTAG
TACAGACGG
TCGCAGCCC
TCCCGGGAC
TCCGTGCGA
GTAGAGGGC
CTCAGACGT
AACGGGTAT
TACAGGTCT

Consensus string: T C C G G G C G C
```



4.1.3) Gibbs sampler which runs 100000 iterations

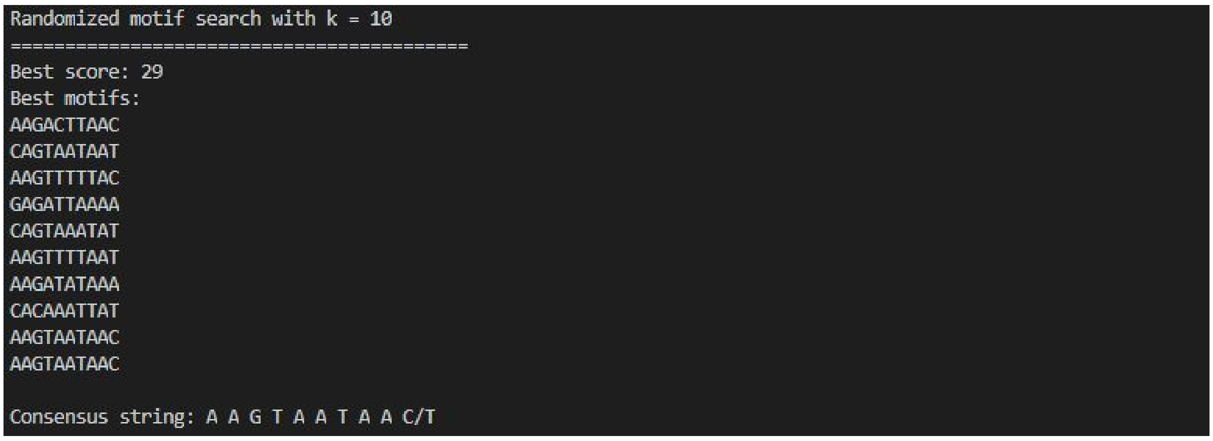


```
Gibbs sampler with k = 9
==========================================
Best score: 19
Best motifs:
TACTCATCC
CGTCAGTAC
TATCGTTGG
CTCGCGAAC
TATCATTGC
TAGGAATCC
TTGCATTCC
GACCATAGC
CAACATTGG
AGTAATAAC

Consensus string: T A T C A T T G C
```

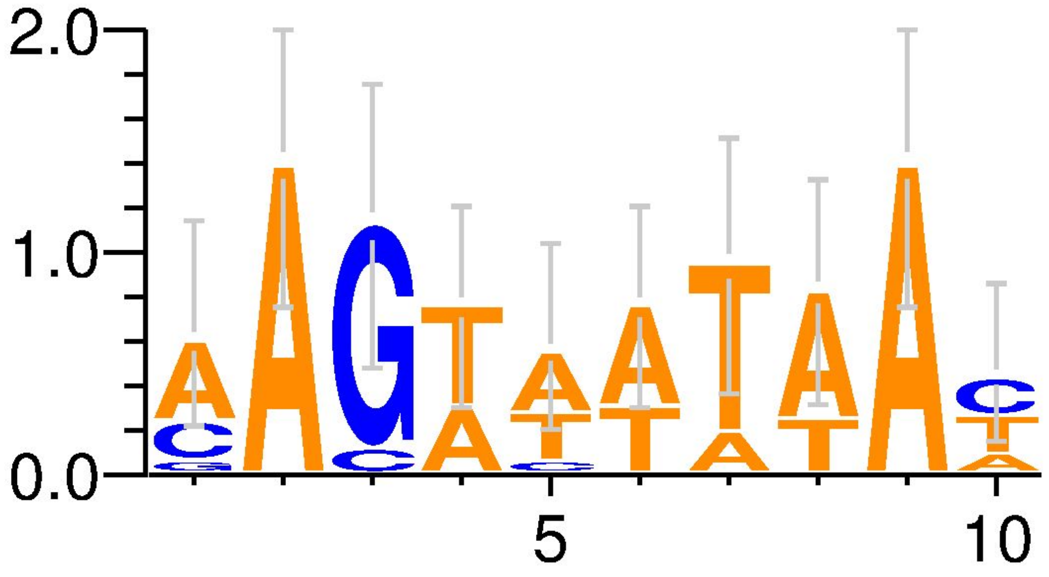


4.2) Run algorithms with motif length 10

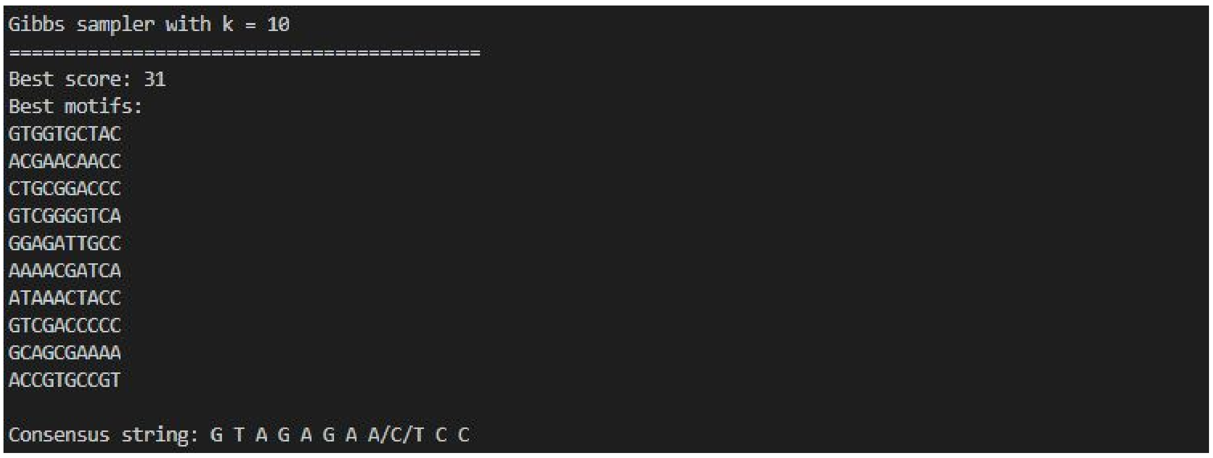
4.2.1) Randomized motif search



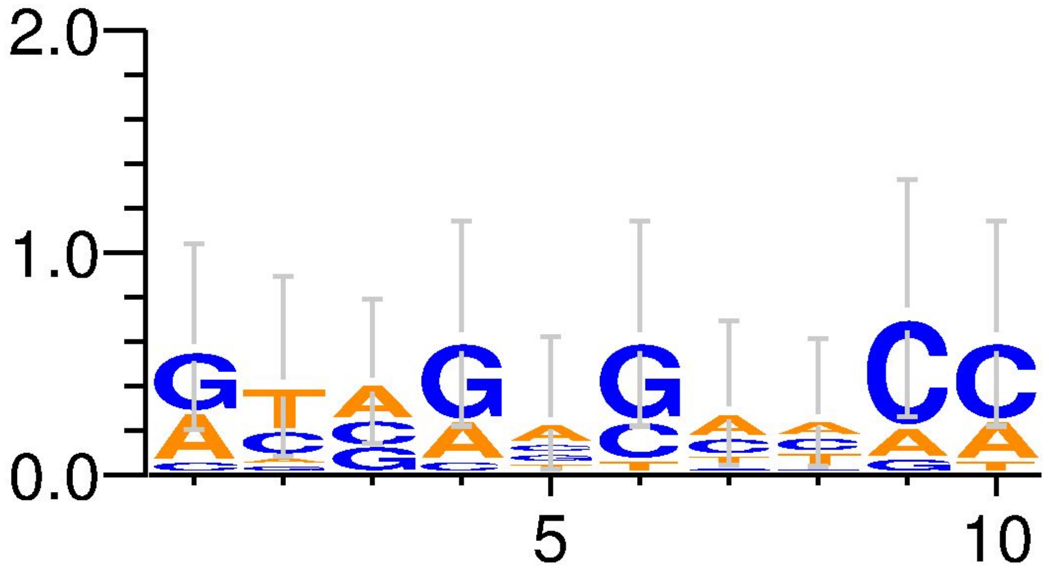
```
Randomized motif search with k = 10
==========================================
Best score: 29
Best motifs:
AAGACTTAAC
CAGTAATAAT
AAGTTTTTAC
GAGATTAAAA
CAGTAAATAT
AAGTTTTAAT
AAGATATAAA
CACAAATTAT
AAGTAATAAC
AAGTAATAAC

Consensus string: A A G T A A T A A C/T
```



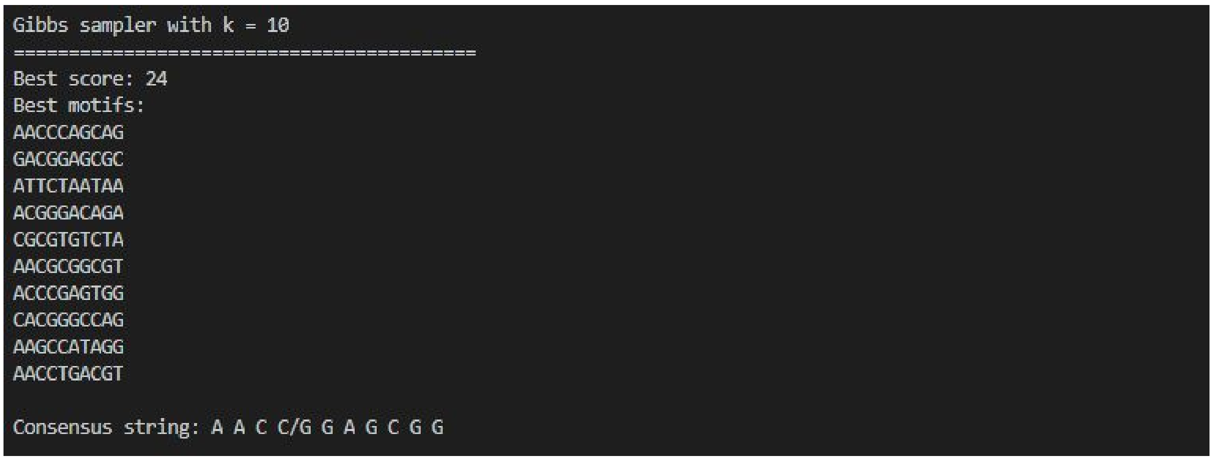
4.2.2) Gibbs sampler which check every 50 iterations



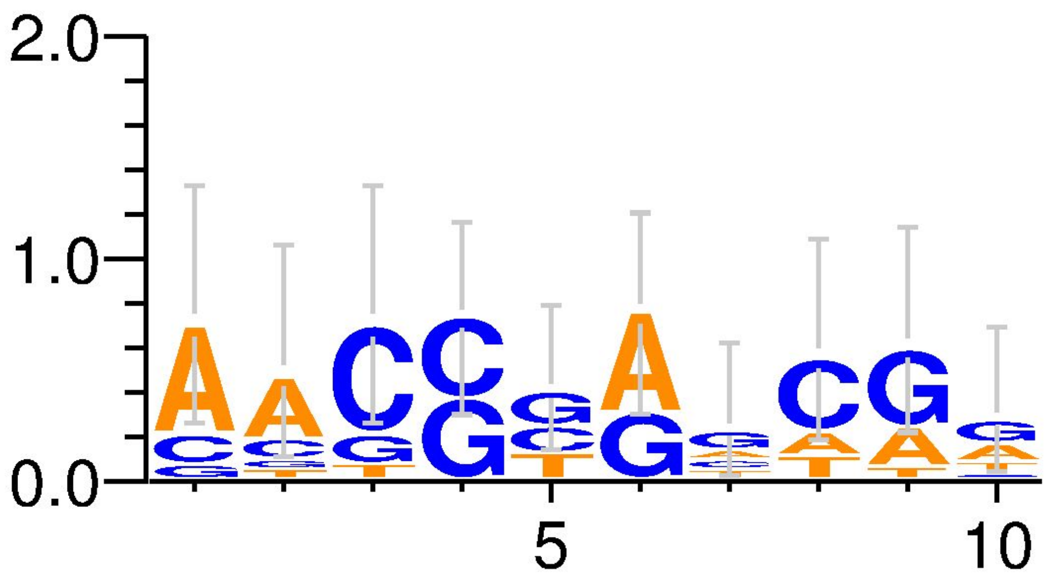
```
Gibbs sampler with k = 10
==========================================
Best score: 31
Best motifs:
GTGGTGCTAC
ACGAACAACC
CTGCGGACCC
GTCGGGGTCA
GGAGATTGCC
AAAACGATCA
ATAAACTACC
GTCGACCCCC
GCAGCGAAAA
ACCGTGCCGT

Consensus string: G T A G A G A A/C/T C C
```



4.2.3) Gibbs sampler which runs 100000 iterations

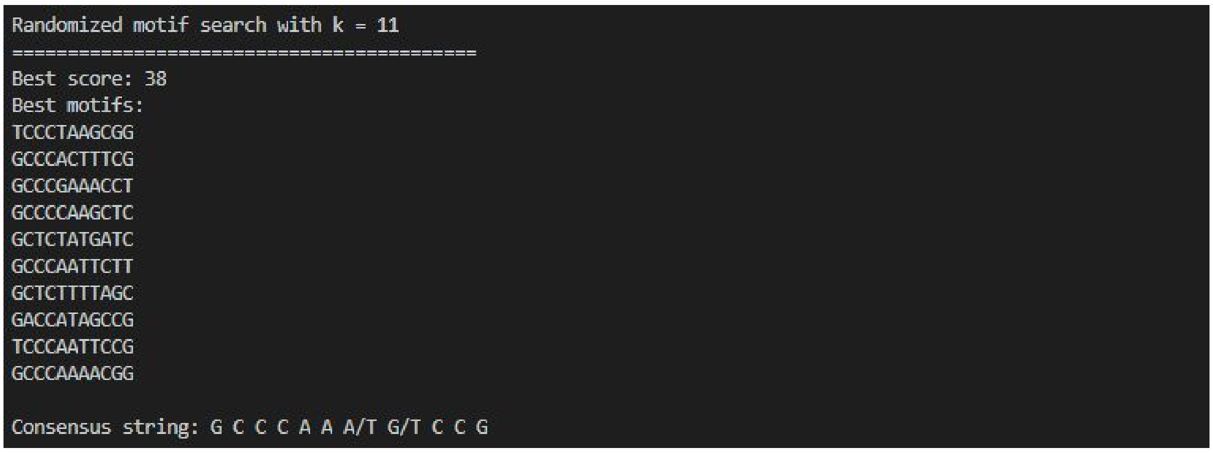


```
Gibbs sampler with k = 10
==========================================
Best score: 24
Best motifs:
AACCCAGCAG
GACGGAGCGC
ATTCTAATAA
ACGGGACAGA
CGCGTGTCTA
AACGCGGCGT
ACCCGAGTGG
CACGGGCCAG
AAGCCATAGG
AACCTGACGT

Consensus string: A A C C/G G A G C G G
```

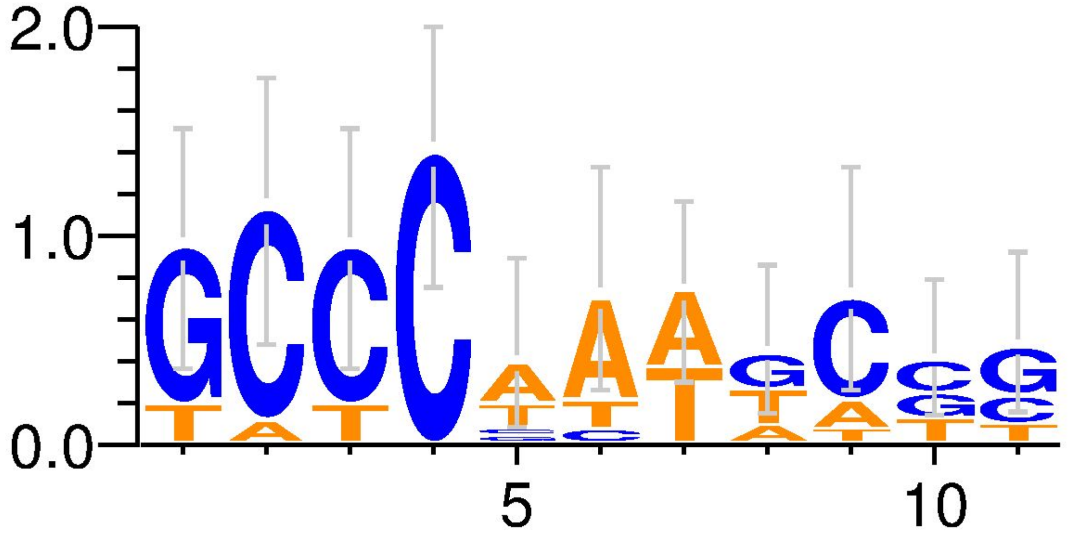


4.3) Run algorithms with motif length 11

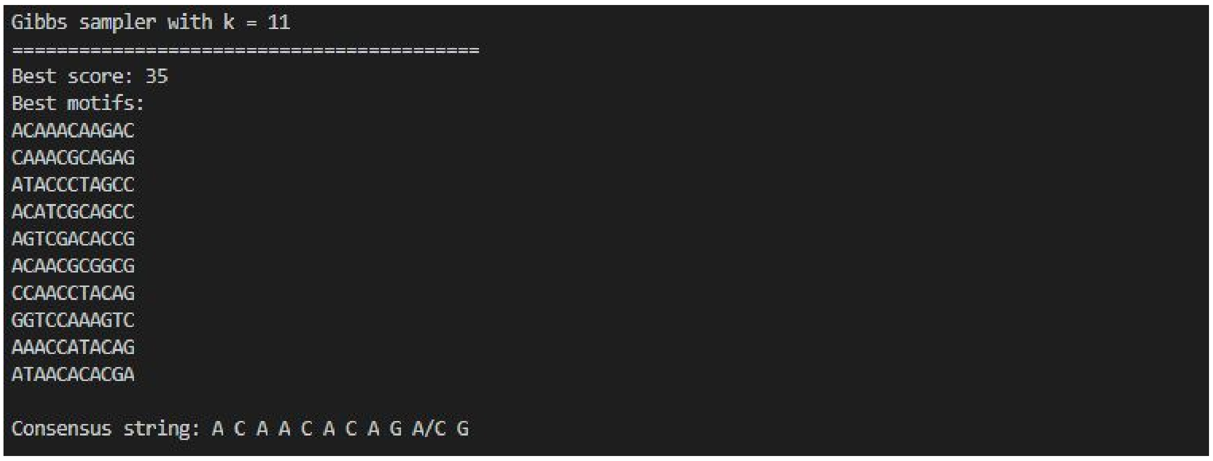
4.3.1) Randomized motif search



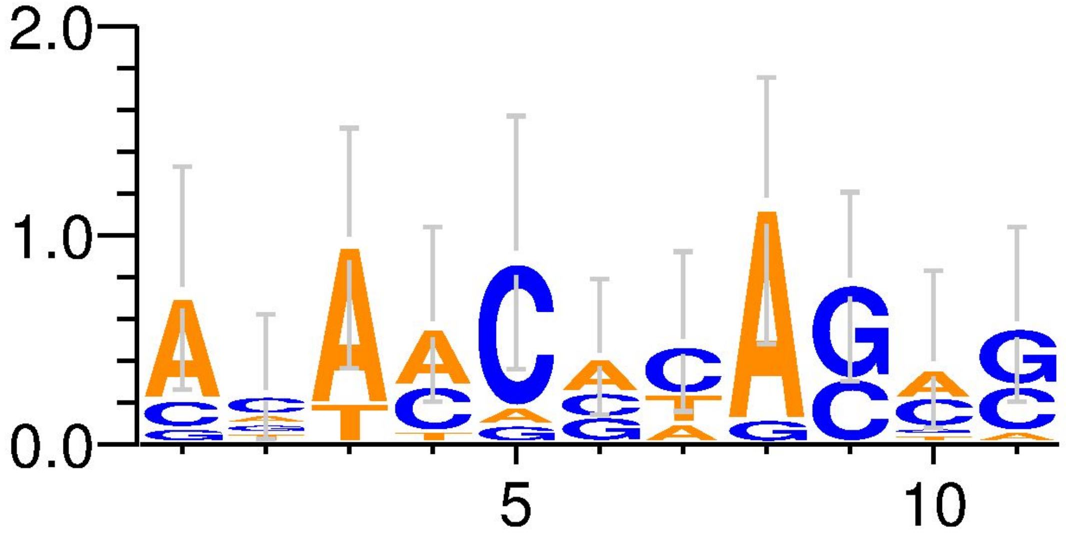
```
Randomized motif search with k = 11
==========================================
Best score: 38
Best motifs:
TCCCTAAGCGG
GCCCACTTTCG
GCCCGAAACCT
GCCCCAAGCTC
GCTCTATGATC
GCCCAATTCTT
GCTCTTTTAGC
GACCATAGCCG
TCCCAATTCCG
GCCCAAAACGG

Consensus string: G C C C A A A/T G/T C C G
```



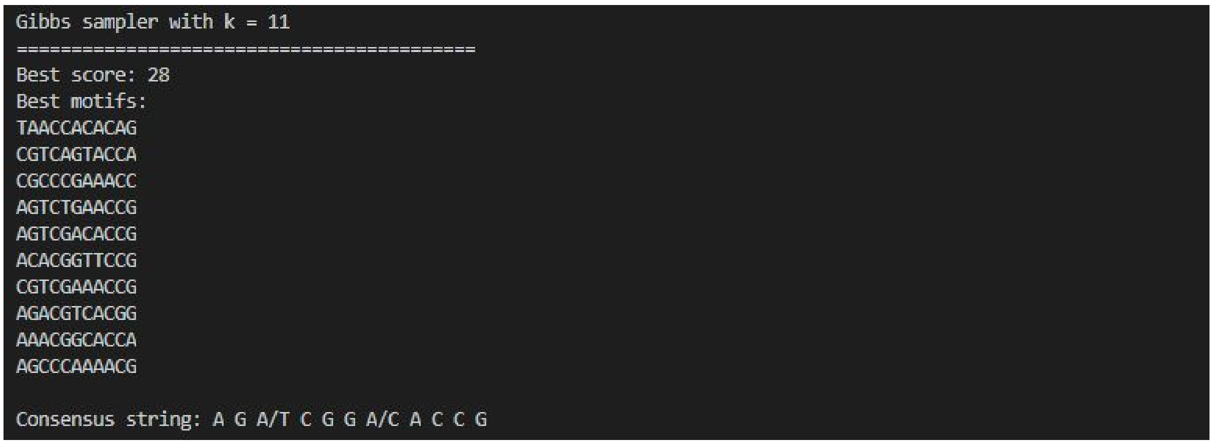
4.3.2) Gibbs sampler which check every 50 iterations



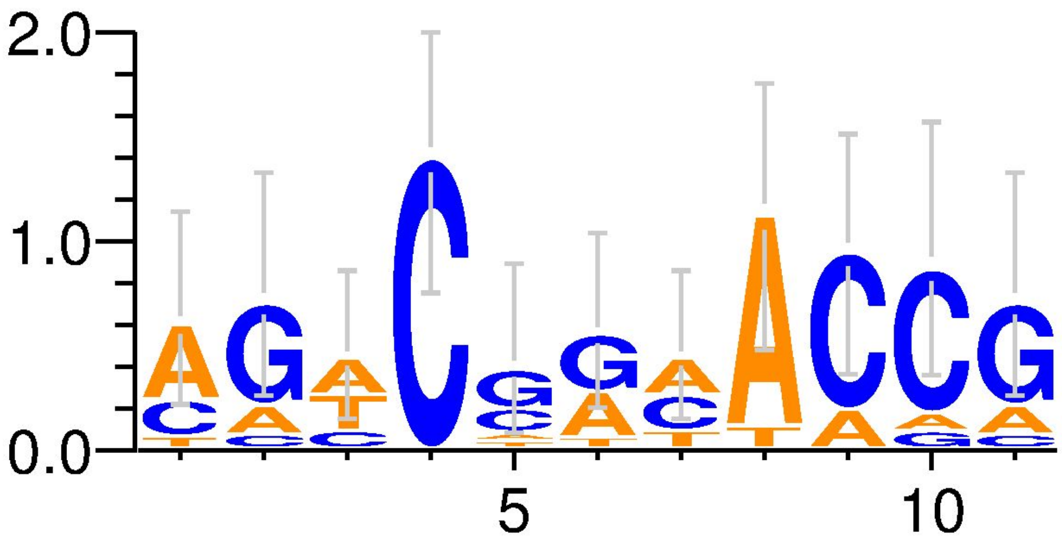
```
Gibbs sampler with k = 11
=========================================
Best score: 35
Best motifs:
ACAAACAAGAC
CAAACGCAGAG
ATACCCTAGCC
ACATCGCAGCC
AGTCGACACCG
ACAACGCGGCG
CCAACCTACAG
GGTCCAAAGTC
AAACCATACAG
ATAACACACGA

Consensus string: A C A A C A C A G A/C G
```



4.3.3) Gibbs sampler which runs 100000 iterations



```
Gibbs sampler with k = 11
==========================================
Best score: 28
Best motifs:
TAACCACACAG
CGTCAGTACCA
CGCCCGAAACC
AGTCTGAACCG
AGTCGACACCG
ACACGGTTCCG
CGTCGAAACCG
AGACGTCACGG
AAACGGCACCA
AGCCCAAAACG

Consensus string: A G A/T C G G A/C A C C G
```



5-) Conclusion

We have inserted ‘AAAAACCCCC’ into each line of dna sequence with different mutations. We have got these results:

- Randomized motif search
  - Motif Length: 9
  - Best Score: 26
  - Consensus String: T T C T G G T T C

- Gibbs sampler which checks every 50 iterations
  - Motif Length: 9
  - Best Score: 29
  - Consensus String: T C C G G G C G C

- Gibbs Sampler which runs 100000 iterations
  - Motif Length: 9
  - Best Score: 19
  - Consensus String: T A T C A T T G C

- Randomized motif search
  - Motif Length: 10
  - Best Score: 29
  - Consensus String: A A G T A A T A A C/T

- Gibbs sampler which checks every 50 iterations
  - Motif Length: 10
  - Best Score: 31
  - Consensus String: G T A G A G A A/C/T C C

- Gibbs Sampler which runs 100000 iterations
  - Motif Length: 10
  - Best Score: 24
  - Consensus String: A A C C/G G A G C G G

- Randomized motif search
  - Motif Length: 11
  - Best Score: 38
  - Consensus String: G C C C A A A/T G/T C C G

- Gibbs sampler which checks every 50 iterations
  - Motif Length: 11
  - Best Score: 35
  - Consensus String: A C A A C A C A G A/C G

- Gibbs Sampler which runs 100000 iterations
  - Motif Length: 11
  - Best Score: 28
  - Consensus String: A G A/T C G G A/C A C C G

Based on the mutations and randomly selected positions, the algorithm finds different consensus strings with different score. It may not find inserted motif, most of the time it does not find.

We can easily observe that randomized motif search and gibbs sampler which check every 50 iterations find approximately same score but different consensus strings. But, gibbs sampler which runs 100000 iterations find very good score.

We have run the code a few times, consensus strings are changing but scores does not change very much.