CSE 4065
Computational Genomics

Programming
Assignment #2

150115062 - Nurcihane KÖROĞLU

150114022 - Oğuzhan BÖLÜKBAŞ

Randomized Motif Search

```
RandomizedMotifSearch(Dna, k, t)
    randomly select k-mers Motifs = (Motif_1, …, Motif_t) in each string from Dna
    BestMotifs ← Motifs
    while forever
        Profile ← Profile(Motifs)
        Motifs ← Motifs(Profile, Dna)
        if Score(Motifs) < Score(BestMotifs)
            BestMotifs ← Motifs
        else
            return BestMotifs
```

Randomized algorithms may be nonintuitive because they lack the control of traditional algorithms. Some randomized algorithms are Las Vegas algorithms, which deliver solutions that are guaranteed to be exact, even though they rely on making random decisions. Yet most randomized algorithms, including the motif finding algorithms that we will consider, is Monte Carlo algorithms. These algorithms are not guaranteed to return exact solutions, but they do quickly find approximate solutions. Because of their speed, they can be run many times, allowing us to choose the best approximation from thousands of runs.

In general, we can begin from a collection of randomly chosen k-mers Motifs in Dna, construct PROFILE(Motifs), and use this profile to generate a new collection of k-mers:

MOTIFS(PROFILE(Motifs), Dna)

Our hope is that MOTIFS(PROFILE(Motifs), Dna) has a better score than the original collection of k-mers Motifs. We can then form the profile matrix of these k-mers,

PROFILE(MOTIFS(PROFILE(Motifs), Dna)),

and use it to form the most probable k-mers,

MOTIFS(PROFILE(MOTIFS(PROFILE(Motifs), Dna)), Dna).

We can continue to iterate…

… PROFILE(MOTIFS(PROFILE(MOTIFS(PROFILE(Motifs), Dna)), Dna)) …

for as long as the score of the constructed motifs keeps improving, which is exactly what Randomized Motif Search does. To implement this algorithm, we will need to randomly select the initial collection of k-mers that form the motif matrix Motifs. To do so, we will need a random number generator.

Since a single run of Randomized Motif Search may generate a rather poor set of motifs, bioinformaticians usually run this algorithm thousands of times. On each run, they begin from a new randomly selected set of k-mers, selecting the best set of k-mers found in all these runs.

Gibbs Sampler

```
GibbsSampler(Dna, k, t, N)
    randomly select k-mers Motifs = (Motif_1, …, Motif_t) in each string from Dna
    BestMotifs ← Motifs
    for j ← 1 to N
        i ← Random(t)
        Profile ← profile matrix constructed from all strings in Motifs except for Motif_i
        Motif_i ← Profile-randomly generated k-mer in the i-th sequence
        if Score(Motifs) < Score(BestMotifs)
            BestMotifs ← Motifs
    return BestMotifs
```

Like Randomized Motif Search, Gibbs Sampler starts with randomly chosen k-mers in each of t DNA sequences, but it makes a random rather than a deterministic choice at each iteration. It uses randomly selected k-mers Motifs = ($Motif_1$, …, $Motif_t$) to come up with another (hopefully better scoring) set of k-mers. In contrast with Randomized Motif Search, which deterministically defines new motifs as:

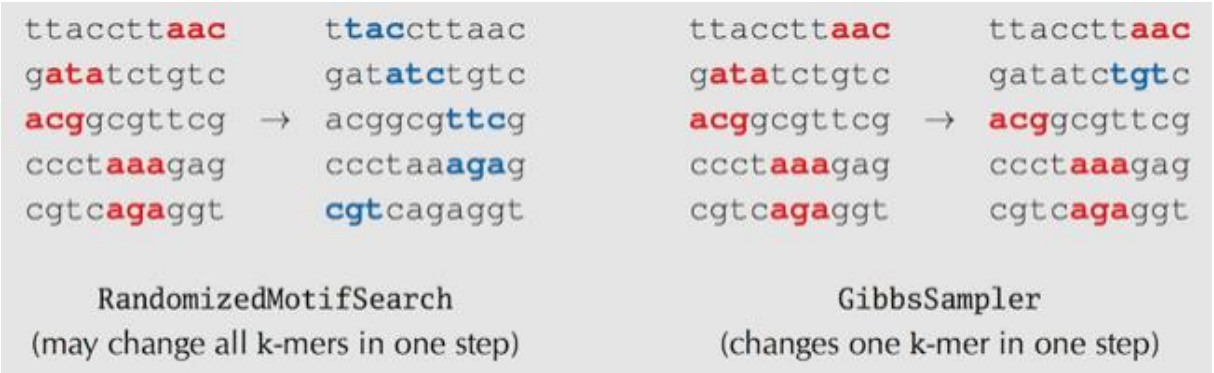
MOTIFS (PROFILE (Motifs), Dna)

Gibbs Sampler randomly selects an integer i between 1 and t and then randomly changes a single k-mer Motif. To describe how Gibbs Sampler updates Motifs, we will need a slightly more advanced random number generator. Given a probability distribution (p_1, ... , p_n), this random number generator, denoted RANDOM(p_1, ... , p_n), models an n-sided biased die and returns integer i with probability p_i.

Although Gibbs Sampler performs well in many cases, it may converge to a suboptimal solution, particularly for difficult search problems with elusive motifs. A local optimum is a solution that is optimal within a small neighbouring set of solutions, which is in contrast to a global optimum, or the optimal solution among all possible solutions. Since Gibbs Sampler explores just a small subset of solutions, it may "get stuck" in a local optimum. For this reason, similarly to Randomized Motif Search, it should be run many times with the hope that one of these runs will produce the best-scoring.

Comparison Between Randomized Motif Search and Gibbs Sampler

1. Randomized Motif Search may change all t strings in Motifs in a single iteration. This strategy may prove reckless, since some correct motifs may potentially be discarded at the next iteration. Gibbs Sampler is a more cautious iterative algorithm that discards a single k-mer from the current set of motifs at each iteration and decides to either keep it or replace it with a new one. This algorithm thus moves with more caution in the space of all motifs, as illustrated below:



2. Randomized motif search can be run for a larger number of iterations to discover better and better motifs. It can also find good solutions for larger values of k.
3. Gibbs Sampler continue until the score of the algorithms no longer improve. For example, we check our score every 50 iterations. If it seems like that the score remains the same for the last 50 iterations, then the algorithm stops to run.

Project Implementation

In this project, we have developed our program with methods. Required operations done in separate methods and their results passed to use to different methods. We have developed randomized motifs search as shown below:

```
# To run Randomized Motif Search Algorithm
def randomized_motif_search(dna_strings, k):
    best_motifs = randomly_choose(dna_strings, k)  # To get randomly p
    old_score = compute_score(best_motifs, k)  # To calculate score of
    while True:
        profile = construct_profile(best_motifs, k)  # To construct pr
        probability = compute_prob(dna_strings, k, profile)  # To calc
        motifs = construct_random_motifs(dna_strings, probability, k)
        new_score = compute_score(motifs, k)  # To calculate score of
        if(new_score < old_score):
            print new_score
            best_motifs = motifs
            old_score = new_score
        else:
            profile = construct_profile(best_motifs, k)  # To construc
            return find_consensus(profile, k), new_score
```

Firstly, the program randomly generates 10 different motifs from 10 different DNA strings which is taken from “input.txt” input file. Then, calculates Score(Motifs) of the randomly generated motifs. The program enters infinite while loop. Then constructs profile matrix of motifs and calculates probability of every k-mers in 500 length 10 different DNA strings one by one and record the result of probability calculation into probability matrix. New motifs are generated using this probability. The program compares each probability and find the highest to find the most probable motifs for every DNA string. After constructing new motifs, it calculates Score(Motifs) and compares the new score with older score result of motifs. If the new score is less than previous one, best motifs are updated, otherwise remains same. If continues until new score is not less than old score.

For Gibbs Sampler(GS) Algorithm, we have designed nearly same as Randomized Motif Search(RMS). Algorithm of GS is shown below:

```
# To run Gibbs Sampler Algorithm
def gibbs_sampler(dna_strings, k):
    count_50 = 0
    best_motifs = randomly_choose(dna_strings, k)  # To get randomly produced motifs
    old_score = compute_score(best_motifs, k)  # To calculate score of the motifs
    while(count_50 != 50):
        profile = construct_profile(best_motifs, k)  # To construct profile matrix of
        probability = compute_prob(dna_strings, k, profile)  # To calculate occurence
        new_motifs = construct_gibbs_motifs(dna_strings, best_motifs, probability, k)
        new_score = compute_score(new_motifs, k)  # To calculate score of the motifs
        if(new_score < old_score):
            print new_score
            best_motifs = new_motifs
            old_score = new_score
        if (old_score == new_score):    count_50 = count_50 + 1
        else:                           count_50 = 0
    profile = construct_profile(best_motifs, k)  # To construct profile matrix of moti
    return find_consensus(profile, k), old_score
```

If we mention only differences from RMS, firstly we can say that execution of GS algorithm ends when Score(Motifs) in last 50 iterations are same. Secondly, we construct profile matrix of motifs with 9 motifs, not 10 as RMS because one motif is supposed as it will be deleted in next iteration, so we do not consider it. Thirdly, we calculate probability of each k-mers in one DNA string not 10 DNA strings. The considered string is which we suppose to removed in next steps. We search next motif with using this probability and find it with biased random number. Then we insert it into motifs instead of deleted one, then the algorithm does same steps until end of its execution.

Outputs of the Algorithms for Different k Values

a. Consensus(Motifs)

- When k = 9

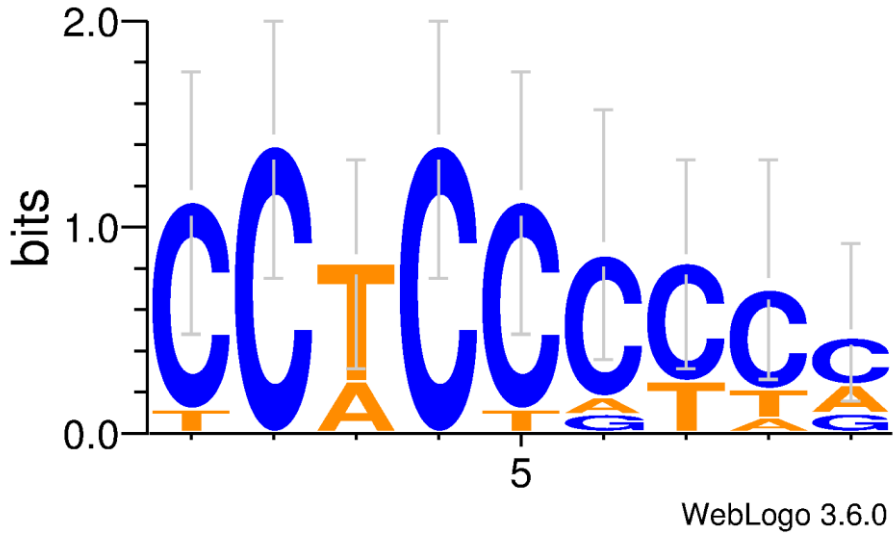


Figure 1: Consensus(Motifs) of Randomized Motif Search

➢ Consensus string of RMS for 9-mers is “CCTCCCCCC”

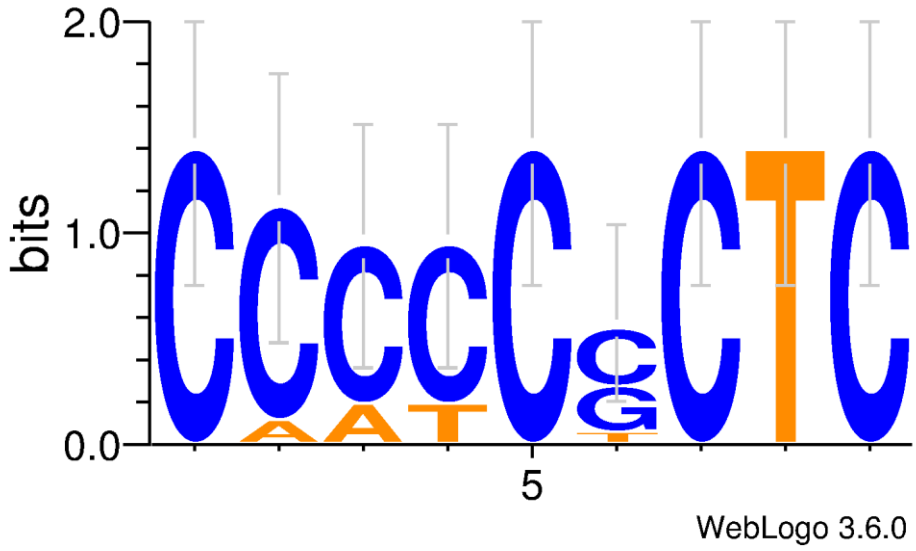


Figure 2: Consensus(Motifs) of Gibbs Sampler

➢ Consensus string of GS for 9-mers is “CCCCCCCTC”

- When k = 10

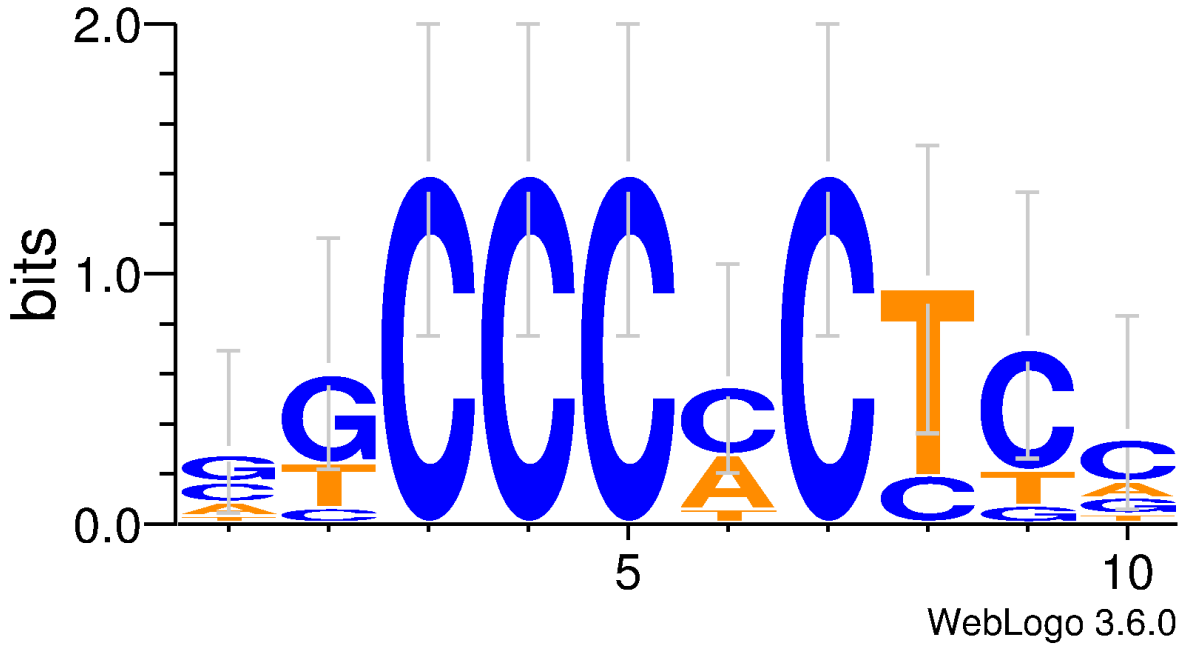


Figure 3: Consensus(Motifs) of Randomized Motif Search

➢ Consensus string of RMS for 10-mers is “GGCCCCCTCC”

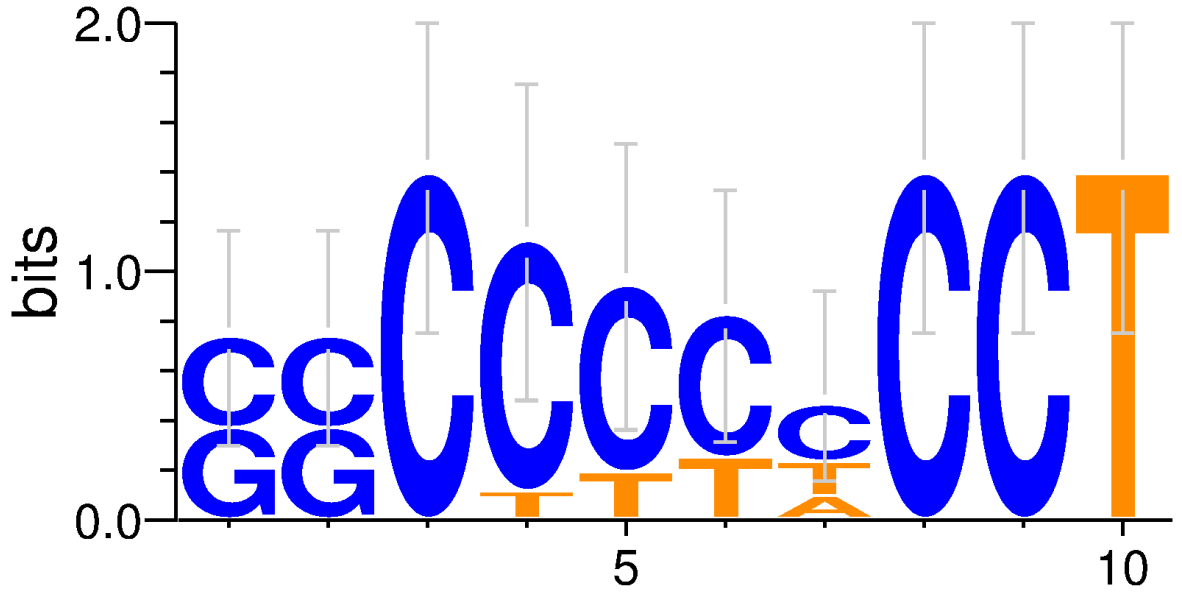


Figure 4: Consensus(Motifs) of Gibbs Sampler

➢ Consensus string of GS for 10-mers is “GGCCCCCCCT”

- When k = 11

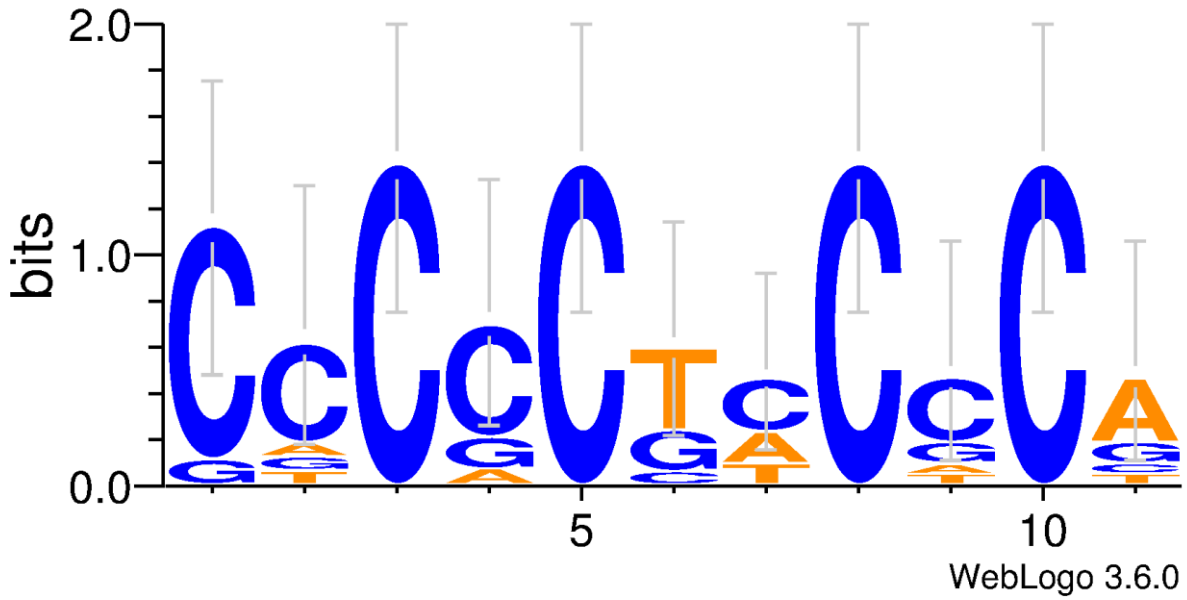


Figure 5: Consensus(Motifs) of Randomized Motif Search

➢ Consensus string of RMS for 11-mers is “CCCCCTCCCCA”

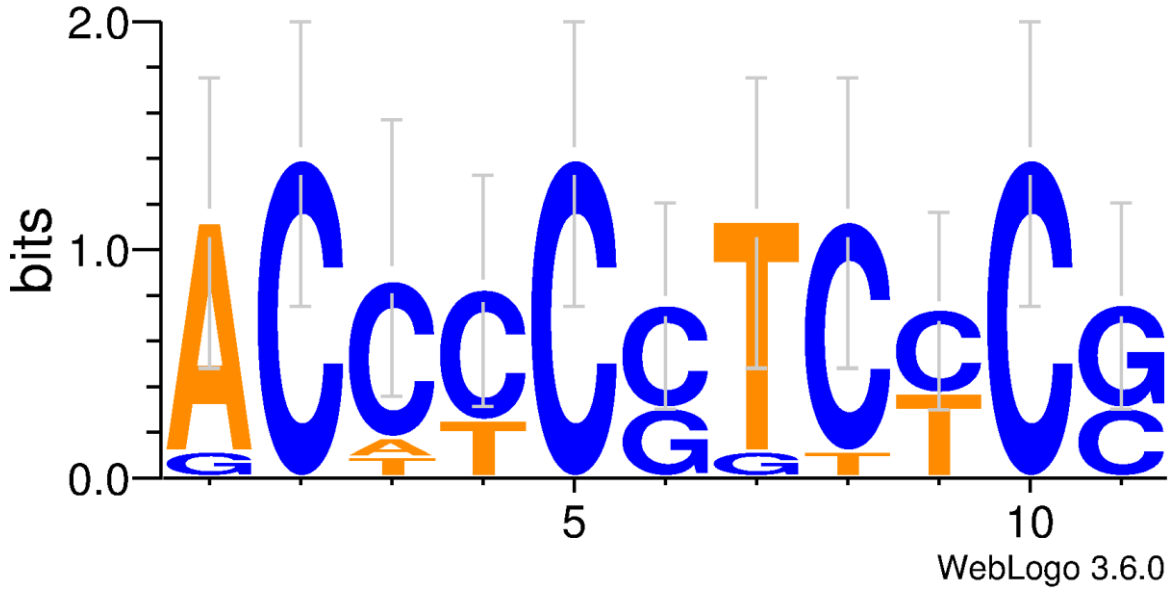
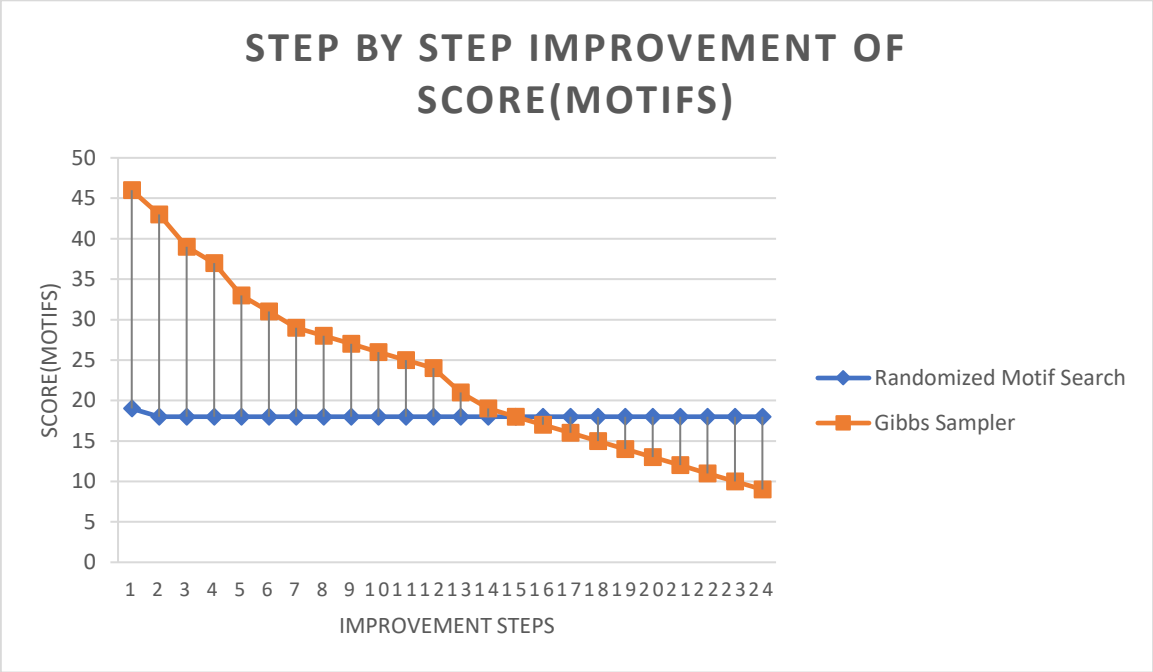


Figure 6: Consensus(Motifs) of Gibbs Sampler

➢ Consensus string of GS for 11-mers is “ACCCCCTCTCG”

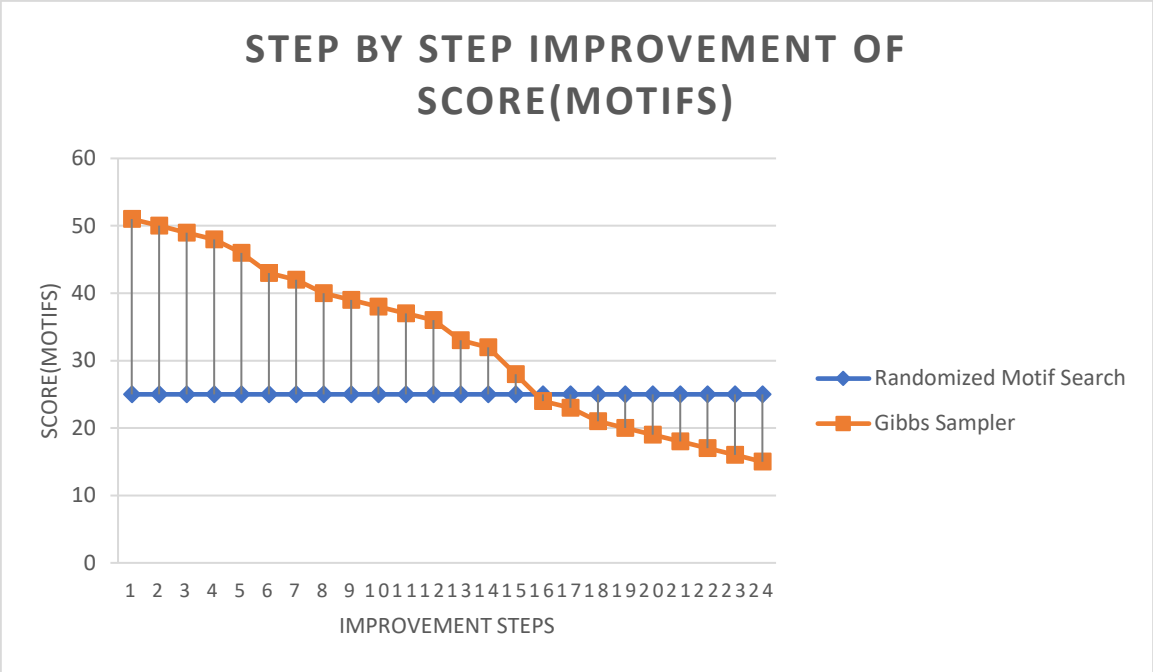
b. Score(Motifs) Graphs

- k = 9



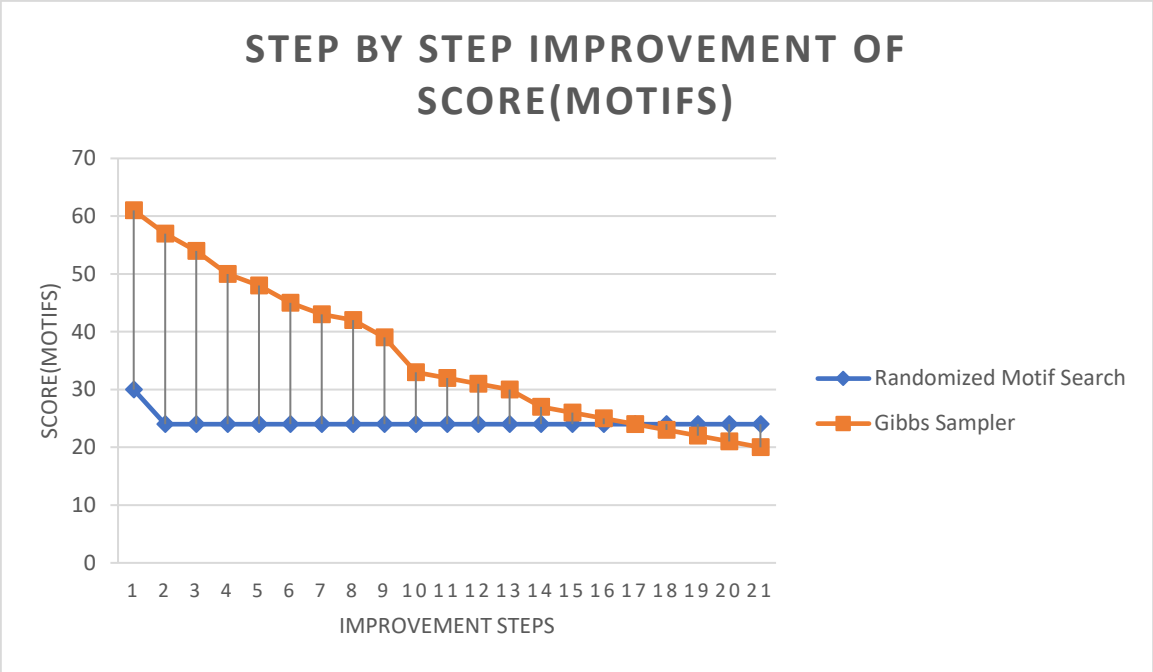
- Randomized Motif Search(RMS) Algorithm find best result with two improvement steps, firstly 19, then 18.
- Gibbs Sampler(GS) Algorithm finds best result with decisive improvement steps, firstly 46, then 9.

- k = 10



- Randomized Motif Search(RMS) Algorithm find best result with only one improvement step, 25.
- Gibbs Sampler(GS) Algorithm finds best result with decisive improvement steps, firstly 51, then 15.

- k = 11



- Randomized Motif Search(RMS) Algorithm find best result with two improvement steps, firstly 34, then 24.
- Gibbs Sampler(GS) Algorithm finds best result with decisive improvement steps, firstly 61, then 20.

Conclusion

Capturing a single implanted motif is often insufficient to steer Randomized Motif Search(RMS) to an optimal solution. Therefore, since the number of starting positions of k-mers is huge, the strategy of randomly selecting motifs is often not successful. The chance that these randomly selected k-mers will be able to guide us to the optimal solution is relatively small.

Although Gibbs Sampler(GS) performs well in many cases, it may converge to a suboptimal solution, particularly for difficult search problems. It explores just a small subset of solutions; it may get stuck in a local optimum. It should be run many times with the hope that one of these runs will produce the best-scoring motifs.

In consideration of the general knowledge about RMS and GS, we have run our experiment with different k-mers like 9-mers, 10-mers and 11-mers. We can see that, RMS can find the solution quickly with less iteration numbers. Whereas, the founded results are not best. Whereas, GS required more execution time and runs with a greater number of iterations, but it finds better results than RMS. The reason of that is GS takes firm steps forward with caution and gives chance to 0 probability strings to be a motif in the Motifs.