T.C.

MARMARA UNIVERSITY

FACULTY of ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

CSE4065 INTRODUCTION TO COMPUTATIONAL GENOMICS PROJECT 1 REPORT

Group Members

150118007 – Sena ALTINTAŞ

150118041 – Mehmet Akif AKKAYA

1. PROJECT DESCRIPTION

In this project, we aimed to find the consensus string using various algorithms that allow searching for motifs. For this, we ran the Randomized Motif Search, Gibbs Sampler and Median String algorithm given to us for 3 different k values and compared these 3 algorithms on the basis of certain parameters.

1. ALGORITHMS

1.1.RANDOMIZED MOTIF SEARCH ALGORITHM

Randomized Motif Search algorithm, which is one of the motif search algorithms, does not guarantee to find the exact solution. It finds the approximate result and does it with as little execution time as possible.

Due to its working speed, it allows to find the best possible result by running it multiple times without wasting time.

```
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)
```

DNA, k value is given as input to the randomized motif search algorithm. First, random k-mers in DNA are created over randomly determined numbers to create a random motif.

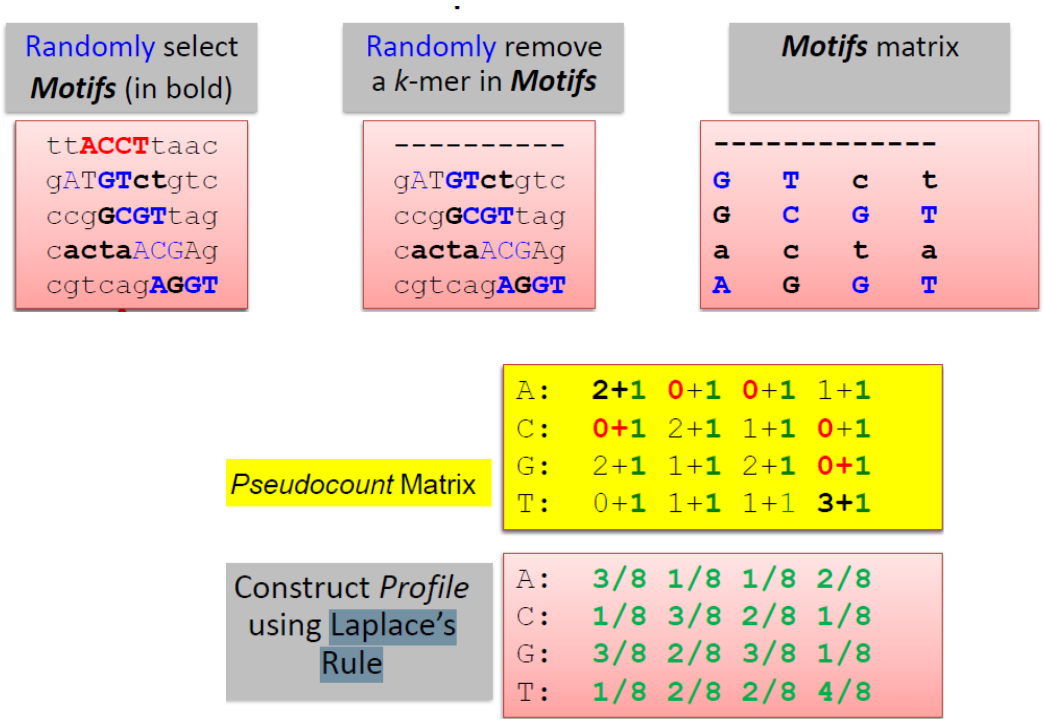
The numbers of A, C, G, T nucleotides in each column and their probabilities are calculated over this motif. This table creates the Profile.

Then, the k-mer with the highest probability in each line is taken over this created Profile and the new motif is selected. Profile calculation is made over this new motif selected.

These processes continue in this way.

While performing the score calculation, the most repeated nucleotide is determined for each column of the motif created in each step. This string creates the consensus string. Using the consensus string, the sum of the different nucleotides in that column is taken. This is done for each column. The found values are summed up and form the score. The general purpose here is to minimize the score.

1.2. GIBBS SAMPLER



In the Gibbs Sampler algorithm, firstly the random k-mer motif is removed from the incoming DNA. Then, the motif is created from randomly selected numbers in the first step from a row of reduced DNA. The number of A, C, G, T nucleotides in each column of the generated motif is calculated. There are nucleotides that have a 0 probability of being found in that column because one row is missing from the motif, and the presence of these 0s will reduce the probability to 0 when calculating the overall profile. To avoid this, 1 is added to each cell when calculating probability. This is called Laplace's Rule. Probability calculation is calculated by adding 1 to the numerator and 4 to the denominator. The probabilities found are calculated as in Randomized Motif Search for the k-mer sequence extracted from the DNA. The k-mer with the highest probability is taken. These steps continue until the lowest score with possible k-mers in all DNA is found.

The score calculation process is the same as in Randomized Motif Search.

1.3.MEDIAN STRING

```
MedianString(Dna, k)
  best-k-mer ← AAA · · · AA
  for each k-mer from AAA · · · AA to TTT · · · TT
    if d(k-mer, Dna) < distance(best-k-mer, Dna)
      best-k-mer ← k-mer
  return(best-k-mer)
```

The median string algorithm takes DNA and k values as input. According to the k value given, it circulates through the entire DNA and outputs the k-mer with the least distance. The distance calculation process in this algorithm finds the difference between the k mer given as input and the DNA.

Runtime: $4^k \cdot n \cdot t \cdot k$ (for *Dna* with t sequences of length n).

Its runtime is calculated as above and is quite large.

2. COMPARISON BETWEEN RANDOMIZED MOTIF SEARCH, GIBBS SAMPLER and MEDIAN STRING ALGORITHM

The randomized motif serach algorithm can run in more iterations. It may give an approximate result, not the actual desired result, but the execution time is quite short compared to other algorithms. The Gibbs sampler algorithm, on the other hand, gives the same result after a certain iteration and cannot improve itself. In the Median String algorithm, the scores are much less compared to the others, they are closer to the desired goal. On the other hand, the execution time is very high compared to other algorithms. While the execution time of the Randomized motif search and Gibbs Sampler algorithm is taken in milliseconds, the execution time of the Median String algorithm is taken over the second unit.

3. PROJECT IMPLEMENTATION

3.1.GENERATE INPUT FILE

While creating the input file, a key of 10-mer length was selected. Each selected key is randomly mutated 4 times. Mutated 10-mers were randomly inserted into 10 DNA sequences consisting of 500 nucleotides in total.

KEY: GTAACGCTCC

3.2. CREATING ALGORITMS

This project was developed using the Java

3.2.1. RANDOMIZED MOTIF SEARCH

```
public static void RandomizedMotifSearch(String key, ArrayList Dna, int k) {
    long start = System.currentTimeMillis();

    x = Dna.size();
    y = k;
    int t = Dna.size();
    int motifScore = 0;
    int oldScore = 0;
    int minScore = -1;
    int counter = 0;
    char[][] motifs = new char[t][k];
    char[][] bestMotifs = new char[t][k];
    double[][] profile = new double[4][k];
    double[][] bestProfile = new double[4][k];

    motifs = CreateRandomMotifs(Dna);
```

First, CurrentTimeMillis() to calculate execution time; method has been used. Then, some constant variables that hold the score and used for the number of iterations are defined. Double two dimensional arrays were created for motif, best motif, profile and best profile.

```
while (true){
    profile = CreateProfile(motifs);
    motifs = CreateMotifs(Dna, profile);
    motifScore = CalculateScore(motifs, key);
    if (oldScore == motifScore){
        counter++;
    } else {counter = 0;}
    oldScore = motifScore;
    if(minScore == -1 || motifScore < minScore){
        minScore = motifScore;
        bestMotifs = motifs.clone();
        bestProfile = profile.clone();
    }
    if (counter >= 50){
        System.out.println("Score: "+CalculateScore(bestMotifs, key));
        System.out.println("----------");
        System.out.println("Best Motif: ");
        for (int i = 0; i<bestMotifs.length; i++){
            for (int j = 0; j<bestMotifs[i].length; j++){
                System.out.print(bestMotifs[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println("----------");
        System.out.println("Consensus String: "+FindConsensus(bestProfile, k));;
```

The algorithm is formed here. Separate methods have been created for all operations such as calculating profile, creating motifs, calculating score, etc. At this stage, each of them is called individually in the necessary areas. The score of the motif created in each iteration is calculated, and if this score is less than the previous score, the new motif and score becomes that. Counter is used to limit the number of iterations and is incremented with each iteration. It will stop when the algorithm runs 50 times. At the end, the algorithm with the least score will create the best motif and print it. Consensus string and score of best motif are also given as output.

3.2.2. GIBBS SAMPLER

```
public static void run(String key, ArrayList Dna, int k) {
    long start = System.currentTimeMillis();
    x = Dna.size();
    y = k;
    int t = Dna.size();
    int motifScore = 0;
    int oldScore = 0;
    int minScore = -1;
    int counter = 0;
    char[][] motifs = new char[t][k];
    double[][] profile = new double[4][k];
    Random rand = new Random();
    for (int i = 0; i<x; i++) {
        int randomNum = rand.nextInt((Dna.get(i).toString().length() - y + 1) + 1);
        randIndex.add(randomNum);
    }
```

currenTimeMillis() to first calculate the execution time; has been used.

Then, some constant variables that hold the score and used for the number of iterations are defined. Double two dimensional arrays were created for motif and profile. Finally, a variable is created using the Random() class to determine a random number. In the for loop, numbers are determined among the numbers in the range of 0-9.

```
while (true){
    motifs = CreateMotifs(Dna);
    int rn = rand.nextInt(x);
    profile = CreateProfile(motifs, rn);
    CalculateProb(rn, String.valueOf(motifs[rn]), profile);
    int currentScore = CalculateScore(motifs,key);
    counter++;
    if (minScore == -1 || minScore > currentScore){
        minScore = currentScore;
        counter = 0;
    }
    if(counter >= 50){
        System.out.println("Score "+minScore);
        System.out.println("---------------------------------");
        System.out.println("Best Motif");
        for (int i = 0; i<motifs.length; i++){
            for (int j = 0; j<motifs[i].length; j++) {
                System.out.print(motifs[i][j]+" ");
            }
            System.out.println();
        }
        System.out.println("---------------------------------");
        System.out.println("Consensus String: "+FindConsensus(profile, k));
```

Where the algorithm works is inside the while(true) loop. Here, methods created separately for each state are called. First, a random motif was created, and then the profile was created. In the CalculateProb(…) method, the probability is calculated over the other lines as if a line was deleted.

The profile calculation has been made for the line that seems to have been deleted from the probability table, and in each new iteration, if the current iteration's score is smaller than the existing one, the new minscore is the score of the current iteration and that motif is determined as the best motif.

As a result of these operations, as in the Randomized motif search algorithm, a consensus string was found by using FindConsensus(profile,k) and given as output.

Since the result repeats itself after a certain step, these procedures were performed up to 50 iterations.

3.2.3. MEDIAN STRING

```
private static int MedianStringAlgorithm(String key, ArrayList dna) {
    int[][] distances = new int[dna.size()][dna.get(0).toString().length() - key.length() + 1];

    for (int i = 0; i < dna.size(); i++){
        for (int j = 0; j < dna.get(i).toString().length() - key.length() + 1; j++){
            for (int a = 0; a<key.length(); a++){
                if (key.toUpperCase().charAt(a) == dna.get(i).toString().toUpperCase().charAt(j+a)){
                }else {
                    distances[i][j] += 1;
                }}}}
    int sum = 0;
    int min = -1;
    for (int i = 0; i<distances.length; i++){
        min = -1;
        for (int j = 0 ; j< distances[i].length; j++){
            if (distances[i][j] < min || min == -1){
                min = distances[i][j];
            }}
        sum += min;}
    if(minSum == -1 || sum < minSum){
        minSum = sum;
        bestPattern = key;
    }
    return sum;}
```

As in the other two algorithms we mentioned in the Median String algorithm, CurrentTimeMillis() to calculate the execution time; We used the method. Then, we compared the strings in the DNA with the key with the for loop. When the equality was not met, the distance was increased by 1. All distances are stored in an array. The pattern with the smallest distance was stored in the variable best pattern.

```
static void printAllKLength(ArrayList dna, char[] set, int k)
{
    int n = set.length;
    printAllKLengthRec(dna, set, "", n, k);
}

static void printAllKLengthRec(ArrayList dna, char[] set, String prefix, int n, int k){

    if (k == 0)
    {
        MedianStringAlgorithm(prefix, dna);
        return;
    }

    for (int i = 0; i < n; ++i)
    {
        String newPrefix = prefix + set[i];
        printAllKLengthRec(dna, set, newPrefix, n, k - 1);
    }
}
```

The printAllKlengthRec() method below is a recursive method. It creates a new prefix in each iteration and runs the above algorithm in base.

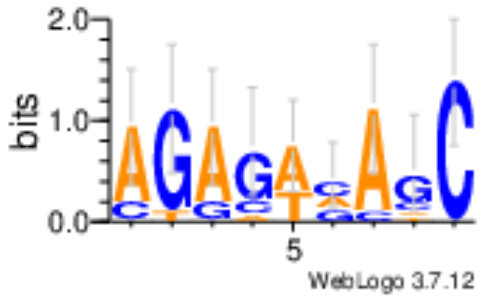
The purpose of this algorithm is to output the pattern with the least distance, the score of this pattern and the execution time.

4. OUTPUT

ITERATIONS	K	RANDOMIZED MOTIF SEARCH		GIBBS SAMPLER	
		SCORE	EXEC. TIME (ms)	SCORE	EXEC. TIME (ms)
1	9	32	49	50	3
	10	35	35	55	8
	11	34	36	61	6
2	9	23	18	53	2
	10	26	17	59	3
	11	33	18	63	3
3	9	29	17	53	1
	10	32	16	59	2
	11	36	19	63	3
4	9	28	16	53	1
	10	32	16	59	1
	11	38	17	63	2
5	9	26	16	53	1
	10	34	16	59	2
	11	39	17	63	1
TOTAL	9	138	116	262	8
	10	159	100	291	16
	11	180	107	313	15
AVARAGE	9	27.6	23.2	52.4	1.6
	10	31.8	20	58.2	3.2
	11	36	21.4	62.6	3
BEST	9	23	16	50	1
	10	26	16	55	1
	11	33	17	61	1

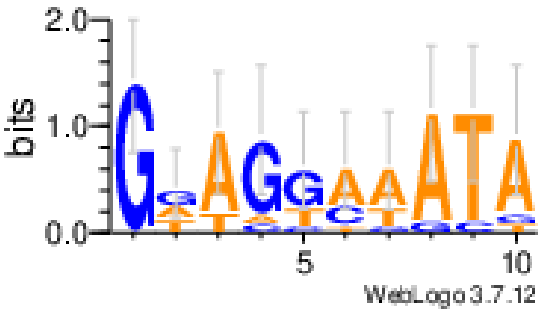
4.1.Output For Randomized Motif Search at Iteration 2

K=9



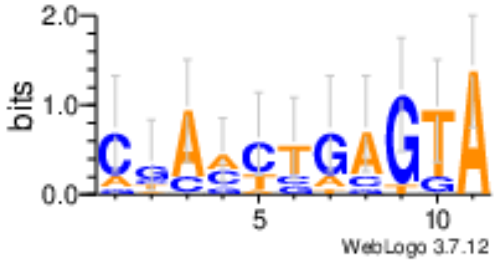
Consensus String: GAAGGAAATA

K=10



Consensus String: AGAGACAGC

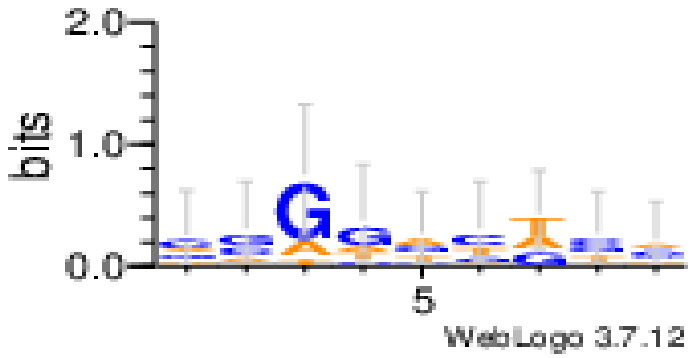
K=11



Consensus String: CTACATGAGTA

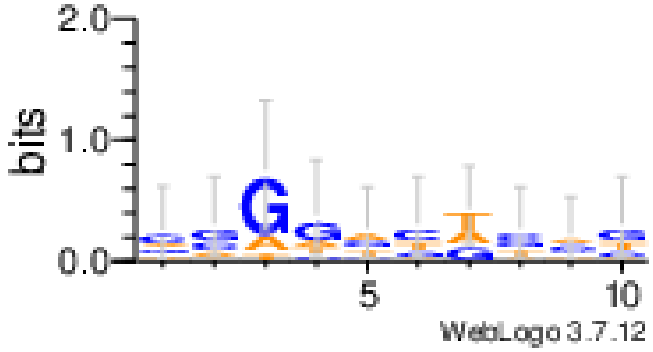
4.2.Output For Gibbs Sampler at Iteration 2

K = 9



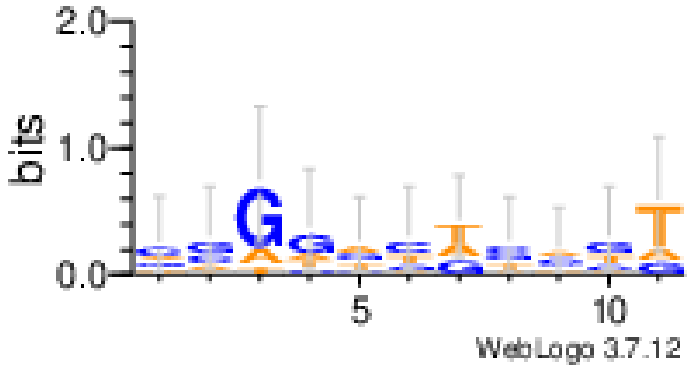
Consensus String: AGGGACTCA

K = 10



Consensus String: CGGAACTCAG

K = 11



Consensus String: CGGGAATCAGT

4.3. Output For Median String

ITERATION	K	MEDIAN STRING	
		SCORE	EXEC. TIME (s)
1	9	14	1998
	10	19	8799
	11	23	40006

K= 9

Pattern: TCGGATGTA

K= 10

Pattern: AGGGCAAACT

K= 11

Pattern: AGGGCAAACTC

In the median string algorithm, we have both low results and the pattern of the 10-mer and 11-mer is the same.

The execution time is quite high, as the median string algorithm creates all the kmers given and searches for the best pattern among them. In addition, the excess of one cycle in the algorithm we have written has increased this time even more.

6. CONCLUSION

When compared to 3 algorithms, the Median String algorithm gives the best result as score. The Gibbs sampler algorithm, shows more consistent results, but cannot heal itself after certain iterations. On the other hand, Median String gives the worst result when compared on the basis of execution time. The Gibbs sampler algorithm works in the least amount of time.