Bangladesh University of Engineering and Technology

Assignment

CSE 463 - Introduction to Bioinformatics

Group 12 -

1905075 Nahida Marzan
1905086 Sushmita Paul
1905093 Soumya Swagata Biswas
1905096 Apurbo Banik Turjo
1905119 Saha Kuljit Shantanu

Contents

1 Data

1.1 Data used in experiments

The data can be found at:
https://github.com/excellencior/CSE-463-Bioinformatics-Assignment/tree/main/Code/input

1.2 Ground Truth Data

The data used as validation can be found at:
https://bioconductor.org/packages/release/data/annotation/html/BSgenome.Hsapiens.UCSC.hg38.html

2 Methods :

In this study, we employed two distinct computational approaches to identify motifs within a given set of DNA sequences: Randomized Motif Search and Gibbs Sampling. Both methods aim to discover common patterns or sequences, known as motifs, which are believed to play significant roles in regulatory functions and other biological processes.

2.1 Randomized Motif Search :

The Randomized Motif Search algorithm begins by randomly selecting k-mers from each sequence in the dataset. It then iteratively refines these motifs by constructing a profile matrix based on the current motifs and using this profile to guide the selection of new, potentially more representative k-mers from each sequence. The algorithm terminates when no improvement in the motif score is observed, returning the best set of motifs found.

2.1.1 Pseudocode :

```
RandomizedMotifSearch(Dna, k, t)
    randomly select k-mers Motifs = (Motif_1, Motif_2, ..., 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
```

Listing 1: Pseudocode for Randomized Motif search

This approach is characterized by its stochastic nature, allowing it to escape local optima by continually sampling the space of possible motifs. However, its performance can be highly dependent on the initial random motifs selected.

2.2 Gibbs Sampling :

Gibbs Sampling, another stochastic technique, offers a more nuanced approach by iteratively updating the set of motifs. In each iteration, it randomly selects one sequence and replaces its current motif with a new one, chosen based on a probabilistic model that considers the motifs in all other sequences. This process allows the algorithm to explore the motif space more thoroughly by making one sequence at a time subject to change.

2.2.1 Pseudocode :

```
GIBBSSAMPLER(Dna, k, t, N)
    randomly select k-mers Motifs = (Motif_1, Motif_2, ..., Motif_t) in each string from Dna
    BestMotifs ← Motifs
    for j ← 1 to N
        i ← RANDOM(t)
```

```
        Profile ← profile matrix formed 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
```

Listing 2: Pseudocode for Gibbs Sampling

Gibbs Sampling's strength lies in its ability to more effectively navigate the search space by considering the impact of individual sequences on the overall motif model. This can potentially lead to a more accurate and representative discovery of motifs.

2.3 Python code implementation

In this section, we delve into the practical application of the motif finding algorithms discussed earlier—Randomized Motif Search and Gibbs Sampling—through their implementation in Python. The code segments provided below encapsulate the core logic and sub-methods integral to each algorithm's operation, facilitating the identification of conserved motifs within DNA sequences.

2.3.1 Basic submethods : Scanning DNA, Random Selctions and Printing motif matrix

At the foundation of our implementation are several basic submethods crucial for both algorithms:

- **Scanning DNA:** The `read_file` function is designed to ingest a file containing DNA sequences and return a list of these sequences, each stripped of newline characters for seamless processing.
- **Random Selections:** The `RandomlySelectKmers` function exemplifies the stochastic nature of our approach by randomly selecting k-mers from the provided DNA sequences, setting the initial state for motif exploration. Additionally, the `RandomlySelectKmers` function is employed in Gibbs Sampling to randomly choose a motif for re-sampling, further illustrating the probabilistic underpinnings of our methods.
- **Printing Motif Matrix:** The `print_mat` function allows for the visualization of the motif matrix, enabling an intuitive understanding of the motifs identified by the algorithms

```python

import random
import math

# Variable declarations
k = 100
max_iterations = 1000
restart_threshold = 10  # Number of iterations before restart

def print_mat(mat):
    for x in mat:
        print(x)

def read_file(filename):
    with open(filename, 'r') as file:
        lines = [line.strip() for line in file.readlines()]
        # To get rid of the trailing \n
    return lines

def RandomlySelectKmers(dna, k):
    t = len(dna)
    kmer_list = []
    for i in range(t):
        start = random.randint(0, len(dna[i]) - k)
        kmer_list.append(dna[i][start:start + k])
    return kmer_list
```

```

def RandomlySelectElimMotif(dna):
    t = len(dna)
    return random.randint(0, t-1)

```

2.3.2 Creating Profile

This sub-method takes a motif matrix as input and gives the profile string as output Central to both Randomized Motif Search and Gibbs Sampling is the construction of a profile matrix from a given set of motifs, as performed by the **Profile** function. This matrix, reflecting the relative frequency of each nucleotide at every position in the motifs, serves as a probabilistic model for motif occurrence.

```

def Profile(motifs):
    t = len(motifs)
    k = len(motifs[0])
    profile = [[1 for i in range(k)] for j in range(4)]
    # Laplace's Rule of Succession (Init at 1)
    for i in range(t):
        for j in range(k):
            if (motifs[i][j] == 'A'):
                profile[0][j] += 1
            elif (motifs[i][j] == 'C'):
                profile[1][j] += 1
            elif (motifs[i][j] == 'G'):
                profile[2][j] += 1
            elif (motifs[i][j] == 'T'):
                profile[3][j] += 1
    for i in range(4):
        for j in range(k):
            profile[i][j] /= (t + 4)
    return profile

```

2.3.3 Identifying the position of motif from a DNA sequence string

This sub-method takes a DNA sequence as input and gives the best suited motif as output The **MOTIFS** function showcases the algorithmic capability to discern the most probable motif within a DNA sequence, guided by the profile matrix. This function iterates through possible k-mers within the sequence, calculating their likelihood based on the current profile and selecting the k-mer with the highest probability.

```

def MOTIFS(text, k, profile):
    max_prob = -1
    kmer = text[:k]
    for i in range(len(text) - k + 1):
        prob = 1
        for j in range(k):
            if (text[i + j] == 'A'):
                prob *= profile[0][j]
            elif (text[i + j] == 'C'):
                prob *= profile[1][j]
            elif (text[i + j] == 'G'):
                prob *= profile[2][j]
            elif (text[i + j] == 'T'):
                prob *= profile[3][j]

```

```
        if (prob > max_prob):
            max_prob = prob
            kmer = text[i:i + k]
    return kmer

```

2.3.4 Scoring

This score method takes a motif matrix as input and gives the score as output. The scoring method used here is implemented with hamming distance heuristics. The method also enhances random restart mechanism for improvising the performance and minimizing the entropy.

```

def Score(motifs):
    t = len(motifs)
    k = len(motifs[0])
    score = 0
    for j in range(k):
        count = {'A': 1, 'C': 1, 'G': 1, 'T': 1}
        # Laplace's Rule of Succession (Init at 1)
        for i in range(t):
            count[motifs[i][j]] += 1
        entropy = 0
        for nucleotide, nucleotide_count in count.items():
            probability = nucleotide_count / (t + 4)
            # Pseudocount for Laplace's Rule of Succession
            entropy -= probability * math.log2(probability)
        score += 2 - entropy  # Maximized entropy is 2 for 4 nucleotides
    return score

```

2.3.5 The Randomized Motif Search

This subsection introduces the **randomized_motif_search** function, embodying the iterative logic of the Randomized Motif Search algorithm. The function iteratively refines the set of motifs by leveraging the profile matrix to guide the selection of more representative k-mers, with the incorporation of a random restart mechanism to avoid local optima.

```

def randomized_motif_search(dna, k, max_iterations, restart_threshold):
    best_motifs = RandomlySelectKmers(dna, k)
    best_score = Score(best_motifs)
    current_iteration = 0
    iteration_count = 0
    while current_iteration < max_iterations:
        motifs = RandomlySelectKmers(dna, k)
        while True:
            profile = Profile(motifs)
            motifs = ["" for i in range(len(dna))]
            for i in range(len(dna)):
                motifs[i] = MOTIFS(dna[i], k, profile)
                # P - most probable k-mer in the i-th string
            current_score = Score(motifs)
            if current_score < best_score:
                best_motifs = motifs
                best_score = current_score
                iteration_count = 0
            elif iteration_count >= restart_threshold:
```

```
                break  # Perform random restart
            else:
                iteration_count += 1
                motifs = motifs
        current_iteration += 1
    return best_motifs

# Example usage:
dna = read_file("yst04r.txt")
best_motifs = randomized_motif_search(dna, k, max_iterations, restart_threshold)
print_mat(best_motifs)

```

2.3.6 The Gibbs Sampling

Similarly, the GibbsSampler function encapsulates the Gibbs Sampling algorithm, emphasizing the iterative update of individual motifs. This function iteratively selects and updates one motif at a time based on a profile constructed from the remaining motifs, allowing for a nuanced exploration of the motif space.

```

def GibbsSampler(dna, k, N):
    t = len(dna)  # Number of sequences
    n = len(dna[0])  # Length of each sequence
    motifs = RandomlySelectKmers(dna, k)  # Initialize motifs randomly
    best_motifs = motifs[:]  # Initialize the best motifs with the initial random motifs

    for _ in range(N):
        i = random.randint(0, t - 1)  # Randomly select a sequence index i
        motif_i = motifs[i]  # Get the motif of the selected sequence

        # Remove the current motif from the profile
        profile_motifs = motifs[:i] + motifs[i + 1:]

        # Construct profile matrix from the other motifs
        profile = Profile(profile_motifs)

        # Select a new motif for sequence i using the profile
        motif_i = MOTIFS(dna[i], k, profile)

        # Update the motifs list with the new motif for sequence i
        motifs[i] = motif_i

        # Update the best motifs if necessary
        if Score(motifs) < Score(best_motifs):
            best_motifs = motifs[:]

    return best_motifs

# Example usage:
dna = read_file("hm03.txt")
best_motifs = GibbsSampler(dna, k, max_iterations)
print_mat(best_motifs)

```

3 Software

3.1 TOOL NAME: rGADEM

Language : R

The code is written in R terminal provided by the R environment.

3.1.1 Software Link

The tool can be found at: https://bioconductor.org/packages/release/bioc/html/rGADEM.html

3.1.2 Commands to Run

```r
pwd <- "D:/rgadem" # Set the working directory
fastaFileName <- "result.fasta"
# Define the filename for the FASTA file
FastaFile <- file.path(pwd, fastaFileName)
# Create the file path to the FASTA file
Sequences <- readDNAStringSet(FastaFile, "fasta")
# Read sequences from the FASTA file
gadem <- GADEM(Sequences, verbose = 1)
# Execute the GADEM algorithm with the sequences
motifs <- gadem@motifList
# Extract the motifs from the GADEM object
print(motifs) # Print the motifs
nMotifs(gadem) # Count the number of motifs
consensus(gadem) # Find the consensus motif
```

3.1.3 Explanation

```
- `pwd <- "D:/rgadem"`:
This line sets the working directory to "D:/rgadem".

- `fastaFileName <- "result.fasta"`:
Here, the filename for the FASTA file is defined as "result.fasta".

- `FastaFile <- file.path(pwd, fastaFileName)`:
This line creates the file path to the FASTA file by combining the working directory
    path (`pwd`) and the FASTA filename (`fastaFileName`).

- `Sequences <- readDNAStringSet(FastaFile, "fasta")`:
Reads sequences from the FASTA file located at `FastaFile` using the `readDNAStringSet`
    function, specifying that the file format is FASTA.

- `gadem <- GADEM(Sequences, verbose = 1)`:
Executes the GADEM algorithm with the sequences stored in `Sequences`, setting the
    verbosity level to 1 (i.e., printing detailed output).

- `motifs <- gadem@motifList`:
Extracts the motifs identified by the GADEM algorithm from the GADEM object `gadem`.

- `print(motifs)`:
Prints the extracted motifs.

- `nMotifs(gadem)`:
Counts the number of motifs identified by the GADEM algorithm.

- `consensus(gadem)`:
Finds the consensus motif among the identified motifs using the GADEM algorithm.
```

Bioconductor biostrings package along with rGADEM package is used as the processing unit of FASTA files for motif discovery from the given data.

3.1.4 Sample output

```
* Start C Programm *
================================================================================
input sequence file:
number of sequences and average length:                          10 1500.0
Use pgf method to approximate llr null distribution
parameters estimated from sequences in:

number of GA generations & population size:                      5 100

PWM score p-value cutoff for binding site declaration:           2.000000e-04
ln(E-value) cutoff for motif declaration:                        0.000000

number of EM steps:                                              40
minimal no. sites considered for a motif:                        2

[a,c,g,t] frequencies in input data:                             0.268596 0.231404
    0.231404 0.268596
================================================================================
* Running an unseeded analysis *
GADEM cycle  1: enumerate and count k-mers... top 3  4, 5-mers: 16 40 34
Done.
Initializing GA... Done.
GADEM cycle[  1] generation[  1] number of unique motif: 1
   spacedDyad: caganntctgt          motifConsensus: CAGCrCTCTGT           1.00 fitness:
      -9.49

GADEM cycle[  1] generation[  2] number of unique motif: 1
   spacedDyad: caganntctgt          motifConsensus: CAGCrCTCTGT           1.00 fitness:
      -9.49

GADEM cycle[  1] generation[  3] number of unique motif: 1
   spacedDyad: caggantctgt          motifConsensus: CAGCrCTCTGT           0.10 fitness:
      -9.49

GADEM cycle[  1] generation[  4] number of unique motif: 1
   spacedDyad: caggantctgt          motifConsensus: CAGCrCTCTGT           0.10 fitness:
      -9.49

GADEM cycle[  1] generation[  5] number of unique motif: 1
   spacedDyad: caggantctgt          motifConsensus: CAGCrCTCTGT           0.10 fitness:
      -9.49
```

3.1.5 Limitations

- **Algorithm limitations :** The motif detection algorithm may have limitations or biases that result in the identification of more motifs than expected.
- **Overlapping motifs :** Motifs can overlap with each other, especially if they are short or if there are repetitive sequences in the input data. This can lead to multiple motifs being identified within the same sequence region.

The sample output for such a scenerio is given here:

```

seq_list <- list(
    "TTTTTTTTTTTTTAGGAAG",
    "TTTTTTTTTTTTTCCTTGT",
    "ATTTTTTTTTTTTCTTTCT",
    "ATTTTTTTTTCTTTCCTCT",
    "CTTTTTTTTTCTTCCTTCT",
    "GCTTTTTTTTTTTATATAT",
    "CTTTTTTTTTCTTTTCTCT",
    "ATTTTTTTTTTGTTAAAAT",
    "GTTTTTTTCTTTTCCTATT",
    "CTATTTTTTTTTTTTTTTT",
```

```
    "CTTTTTTTTATTTCAAATT",
    "GTTTTGTTCTTTTTGCAAA",
    "ATTTTTTGTTTGTATTCTT",
    "CTTTTTCTCTTTTTTACAG",
    "ATTTTTCTTTTCTTAGTTT",
    "CTTGTTTCTTTTTCTGCAC",
    "CTATTTTTTTCTTCCTTAT",
    "ATATTTTTCTTTTTAATTC",
    "GTTTTTTATTCTTAACTTG"
)
```

3.1.6 Detailed output

It can be observed that rGADEM analysis uses masking of properly motif offset which includes the discovered motifs and also it maybe the reason for duplicate motif finding from the strings.

```

* Start C Programm *
==========================================================================================
input sequence file:
number of sequences and average length:                          11 1000.0
Use pgf method to approximate llr null distribution
parameters estimated from sequences in:

number of GA generations & population size:                      5 100

PWM score p-value cutoff for binding site declaration:           2.000000e-04
ln(E-value) cutoff for motif declaration:                        0.000000

number of EM steps:                                              40
minimal no. sites considered for a motif:                        2

[a,c,g,t] frequencies in input data:                             0.303892 0.196108
    0.196108 0.303892
==========================================================================================
* Running an unseeded analysis *
GADEM cycle  1: enumerate and count k-mers... top 3  4, 5-mers: 6 12 16
Done.
Initializing GA... Done.
GADEM cycle[  1] generation[  1] number of unique motif: 1
   spacedDyad: aagannnaaa           motifConsensus: AAAAAAAAAA            0.60 fitness:
     -15.82

GADEM cycle[  1] generation[  2] number of unique motif: 1
   spacedDyad: aagaannngaa          motifConsensus: AAGAAAAArAA           0.40 fitness:
     -16.72

GADEM cycle[  1] generation[  3] number of unique motif: 1
   spacedDyad: aaaannngaaag         motifConsensus: AAAAAAAAAAAA          1.00 fitness:
     -19.21

GADEM cycle[  1] generation[  4] number of unique motif: 1
   spacedDyad: aaaannngaaag         motifConsensus: AAAAAAAAAAAA          1.00 fitness:
     -19.21

GADEM cycle[  1] generation[  5] number of unique motif: 1
   spacedDyad: aaaannngaaag         motifConsensus: AAAAAAAAAAAA          1.00 fitness:
     -19.21

* Running an unseeded analysis *
GADEM cycle  2: enumerate and count k-mers... top 3  4, 5-mers: 10 6 8
Done.
Initializing GA... Done.
GADEM cycle[  2] generation[  1] number of unique motif: 1
   spacedDyad: tttcnnnnctt          motifConsensus: TTTCCCyCCTT           0.50 fitness:
      12.22

GADEM cycle[  2] generation[  2] number of unique motif: 1
   spacedDyad: ttccnnnnctt          motifConsensus: TTCCCCTCyTT           0.50 fitness:
      12.14

GADEM cycle[  2] generation[  3] number of unique motif: 1
```

```
   spacedDyad: ttccnnnncttt          motifConsensus: TTCCCCTCyTT          0.50 fitness:
       9.37

GADEM cycle[  2] generation[  4] number of unique motif: 1
   spacedDyad: ttccnnnncttt          motifConsensus: TTCCCCTCyTT          0.50 fitness:
       9.37

GADEM cycle[  2] generation[  5] number of unique motif: 1
   spacedDyad: ttccnnnncttt          motifConsensus: TTCCCCTCyTT          0.50 fitness:
       9.37
```

3.2 TOOL NAME : SLiMFinder

Running Process : Online

The tool is run on an online server that takes necessary input files and parameters and outputs the motif matches

3.2.1 Software Link

The tool can be found at: http://www.slimsuite.unsw.edu.au/servers/slimfinder.php

3.2.2 Interface

SLiMFinder Interface

Explanation: Here we set it such that we only take the top 20 ranked motifs of length 20 or less, this means that there can be motifs of lesser length as well.

3.2.3 Generated Output

After running SLiMFinder, we find that there are motifs of different length with different start-positions and end-positions. Here we show a portion of the whole output

Dataset	RunID	Rank	Pattern	Sig	Seq	Start_Pos	End_Pos	Prot_Len	Match	Variant	MisMatch	Desc	PepSeq	PepDesign
slimfinder	24031400014	1	^A.G	1.00	seq3_UNK__seq3	1	3	1500	AGG	^A.G	0		AGG	OK
slimfinder	24031400014	1	^A.G	1.00	seq8_UNK__seq8	1	3	1500	AGG	^A.G	0		AGG	OK
slimfinder	24031400014	1	^A.G	1.00	seq9_UNK__seq9	1	3	1500	AGG	^A.G	0		AGG	OK
slimfinder	24031400014	2	^AGG	1.00	seq3_UNK__seq3	1	3	1500	AGG	^AGG	0		AGG	OK
slimfinder	24031400014	2	^AGG	1.00	seq8_UNK__seq8	1	3	1500	AGG	^AGG	0		AGG	OK
slimfinder	24031400014	2	^AGG	1.00	seq9_UNK__seq9	1	3	1500	AGG	^AGG	0		AGG	OK
slimfinder	24031400014	3	AGG.A	1.00	seq3_UNK__seq3	1	5	1500	AGGTA	AGG.A	0		AGGTA	OK
slimfinder	24031400014	3	AGG.A	1.00	seq4_UNK__seq4	478	482	1500	AGGAA	AGG.A	0		AGGAA	OK
slimfinder	24031400014	3	AGG.A	1.00	seq9_UNK__seq9	1	5	1500	AGGCA	AGG.A	0		AGGCA	OK
slimfinder	24031400014	4	^AG	1.00	seq3_UNK__seq3	1	2	1500	AG	^AG	0		AG	OK
slimfinder	24031400014	4	^AG	1.00	seq8_UNK__seq8	1	2	1500	AG	^AG	0		AG	OK
slimfinder	24031400014	4	^AG	1.00	seq9_UNK__seq9	1	2	1500	AG	^AG	0		AG	OK
slimfinder	24031400014	5	^A.{0,1}G.{0,2}G	1.00	seq3_UNK__seq3	1	3	1500	AGG	^AGG	0		AGG	OK
slimfinder	24031400014	5	^A.{0,1}G.{0,2}G	1.00	seq3_UNK__seq3	1	6	1500	AGGTAG	^A.G..G	0		AGGTAG	OK
slimfinder	24031400014	5	^A.{0,1}G.{0,2}G	1.00	seq8_UNK__seq8	1	3	1500	AGG	^AGG	0		AGG	OK
slimfinder	24031400014	5	^A.{0,1}G.{0,2}G	1.00	seq8_UNK__seq8	1	4	1500	AGGG	^A.GG	0		AGGG	OK

SLiMFinder Result

3.2.4 Motif Generation

According to the consensus, a portion of the matched pairs is shown below. They are sorted by rank and also by match percentage.

```
slimfinder: 10 Seq, 10 UPC, 20 Sig SLiMs in 8 Clouds

Cloud 1 (1 SLiMs):      ^A.G    (p = 1.00)
3 UPC   3 Seq:  seq3_UNK__seq3  seq8_UNK__seq8  seq9_UNK__seq9

SLiMMaker Consensus: AGG
- SLiM matches 3 of 3 sequences (100.0%).

-AGG
-AGG
-AGG

Cloud 2 (1 SLiMs):      ^AGG    (p = 1.00)
3 UPC   3 Seq:  seq3_UNK__seq3  seq8_UNK__seq8  seq9_UNK__seq9

SLiMMaker Consensus: AGG
- SLiM matches 3 of 3 sequences (100.0%).

-AGG
-AGG
-AGG
```

Generated Motif Part 1

```
Cloud 3 (5 SLiMs):      AGG.A   A.G.A   AG..A   AGG     GG.A    (p = 1.00)
4 UPC   4 Seq:  seq3_UNK__seq3  seq4_UNK__seq4  seq8_UNK__seq8  seq9_UNK__seq9

SLiMMaker Consensus: AGG.A
- SLiM matches 5 of 9 sequences (55.6%).

AGGTA
AGGAA
AGGCA
AGGGA
AGGGG
AGGAA
TGGAA
AAGGA
ATGTA

Cloud 4 (1 SLiMs):      ^AG     (p = 1.00)
3 UPC   3 Seq:  seq3_UNK__seq3  seq8_UNK__seq8  seq9_UNK__seq9

SLiMMaker Consensus: AG
- SLiM matches 3 of 3 sequences (100.0%).

-AG
-AG
-AG
```

Generated Motif Part 2

```
Cloud 6 (1 SLiMs):      ^.GG    (p = 1.00)
3 UPC   3 Seq:  seq3_UNK__seq3  seq8_UNK__seq8  seq9_UNK__seq9

SLiMMaker Consensus: AGG
- SLiM matches 3 of 3 sequences (100.0%).

-AGG
-AGG
-AGG

Cloud 7 (9 SLiMs):      TT.G.{0,1}T     TT.{1,2}GT      TTG.{0,1}G      T..GT   TT..G   TT.{0,1}G.{0,1}G        TT.G    TTG     T..G.{0,1}G     (p = 1.00)
3 UPC   3 Seq:  seq3_UNK__seq3  seq4_UNK__seq4  seq7_UNK__seq7

SLiMMaker Consensus: TT[GT]G[GT][GT]
- SLiM matches 4 of 8 sequences (50.0%).

TTGGGT
TTTGGT
TTGTGG
TTGGTT
TTTGTG
GTTTGT
ATTGGG
TGGGTG

Cloud 8 (1 SLiMs):      TGT     (p = 1.00)
3 UPC   3 Seq:  seq4_UNK__seq4  seq7_UNK__seq7  seq8_UNK__seq8

SLiMMaker Consensus: TGT
- SLiM matches 3 of 3 sequences (100.0%).

TGT
TGT
TGT
```

Generated Motif Part 3

```
#Cloud UPC coverage/overlap
Cloud   Dataset 1       2       3       4       5       6       7       8
1:^A.G  0.300   1.000   1.000   0.750   1.000   1.000   1.000   0.333   0.333
2:^AGG  0.300   1.000   1.000   0.750   1.000   1.000   1.000   0.333   0.333
3:AGG.A 0.400   1.000   1.000   1.000   1.000   1.000   1.000   0.667   0.667
4:^AG   0.300   1.000   1.000   0.750   1.000   1.000   1.000   0.333   0.333
5:^A.{0,1}G.{0,2}G      0.300   1.000   1.000   0.750   1.000   1.000   1.000   0.333   0.333
6:^.GG  0.300   1.000   1.000   0.750   1.000   1.000   1.000   0.333   0.333
7:TT.G.{0,1}T   0.300   0.333   0.333   0.500   0.333   0.333   0.333   1.000   0.667
8:TGT   0.300   0.333   0.333   0.500   0.333   0.333   0.333   0.667   1.000
```

Generated Motif Overlap Information

The overlap information's helps us determine what part of the consensus are being matched. Also the generated motifs are being ranked based on their quality and precision.

3.2.5 Portion of Generated Log

#DB	00:00:00	/share/apps/blast+/2.7.1/bin/makeblastdb -in "/srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.slimdb" -out "/srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.slimdb" -title "/srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.slimdb" -parse_seqids -dbtype prot
#SYS	00:00:00	/share/apps/blast+/2.7.1/bin/blastp -query /srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.slimdb -db /srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.slimdb -out /srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.self.blast -seg yes -comp_based_stats F -soft_masking true -evalue 1.000000e-04 -num_descriptions 10 -num_alignments 10
#BLAST	00:00:00	Deleted BLAST results file: /srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.self.blast
#CLUST	00:00:00	10 clusters generated from 10 keys (maxdis=1.00)
#DICT	00:00:00	Made "short" seqName dictionary: 10 seq; 10 keys.
#MST	00:00:00	MST for 10 sequences = 10.000.
#UP	00:00:00	slimfinder: 10 Seq; 10 UPC; 10.000 MST; blaste=1.00e-04, blastcf=F, blastf=T
#MAT	00:00:00	slimfinder GABLAM Distance matrix out to /srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.dis.tdt (text format).
#MIN	00:00:00	AmbOcc 0.05 & 10 UP => AmbOcc 2 UPC.
#MIN	00:00:00	MinOcc 0.05 & 10 UP => MinOcc 3 UPC.
#DIS	00:00:00	Disorder score file: /srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.iushort2.txt
#MASK	00:00:00	seq1_UNK__seq1 masked 1 ordered regions. (1499 X added.)
#MASK	00:00:00	seq2_UNK__seq2 masked 3 ordered regions. (1494 X added.)
#MASK	00:00:00	seq3_UNK__seq3 masked 2 ordered regions. (1482 X added.)
#MASK	00:00:00	seq4_UNK__seq4 masked 10 ordered regions. (1459 X added.)
#MASK	00:00:00	seq5_UNK__seq5 masked 1 ordered regions. (1497 X added.)
#MASK	00:00:00	seq6_UNK__seq6 masked 1 ordered regions. (1500 X added.)
#MASK	00:00:00	seq7_UNK__seq7 masked 4 ordered regions. (1482 X added.)
#MASK	00:00:00	seq8_UNK__seq8 masked 2 ordered regions. (1494 X added.)
#MASK	00:00:00	seq9_UNK__seq9 masked 1 ordered regions. (1493 X added.)
#MASK	00:00:00	seq10_UNK__seq10 masked 1 ordered regions. (1500 X added.)
#MASK	00:00:00	Masked seq4_UNK__seq4 "low complexity" regions. (5 X added.)

Generated Motif log part 1

#~~#	#~+~~+~#	#~~#
#LOG	00:00:00	Activity Log for SLiMSuite V1.11.0: Thu Mar 14 17:52:21 2024
#DIR	00:00:00	Run from directory: /srv/slimsuite/logs/prod
#ARG	00:00:00	Commandline arguments: run ip=129.94.146.29 slimfinder&seqin=url:web.slimsuite.unsw.edu.au/web/input/202403141752.seqin&probcut=1.0&dismask=T&topranks=20&uniprotid=LIG_Dynein_DLC8_1&jobid=24031400014
#VIO	00:00:00	Verbosity: -1; Interactivity: -1.
#PROG	00:00:00	SLiMFinder V5.4.0: Short Linear Motif Finder
#CMD	00:00:00	Full SLiMFinder CmdList: proglog=F iucut=0.2 orthdb=qfo_ref.2019-04.Eukaryota.fas sourcedate=2015-04-15 maxgapx=3 usegopher=T protscores=T megablam=F restab=Search,Hit,Local qassemble=T combinedfas=T treeformats=nwk,text,png runid=24031400014 walltime=2.0 rest=format seqin=slimfinder.seqin probcut=1.0 dismask=T topranks=20 uniprotid=LIG_Dynein_DLC8_1.acc jobid=24031400014 resfile=slimfinder.csv savespace=0 targz=F megaslim=seqin basefile=slimfinder
#DB	00:00:00	/srv/slimsuite/data/orthdb/qfo_ref.2019-04/qfo_ref.2019-04.Eukaryota.fas already formatted. (Force = False).
#DB	00:00:00	/srv/slimsuite/data/orthdb/qfo_ref.2019-04/qfo_ref.2019-04.Eukaryota.fas already formatted. (Force = False).
#ALPH	00:00:00	Alphabet: A C D E F G H I K L M N P Q R S T V W Y
#WARN	00:00:00	NOTE: cloudfix=F. Be wary of ambiguity over-predictions.
#SEQ	00:00:00	10 sequences loaded from /srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.seqin (Format: fas).
#UPC	00:00:00	No UPC file found.
#FAS	00:00:00	10 Sequences output to /srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.slimdb.
#DICT	00:00:00	Made "short" seqName dictionary: 10 seq; 10 keys.
#DB	00:00:00	/share/apps/blast+/2.7.1/bin/makeblastdb -in "/srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.slimdb" -out "/srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.slimdb" -title "/srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.slimdb" -parse_seqids -dbtype prot
#SYS	00:00:00	/share/apps/blast+/2.7.1/bin/blastp -query /srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.slimdb -db /srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.slimdb -out /srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.self.blast -seg yes -comp_based_stats F -soft_masking true -evalue 1.000000e-04 -num_descriptions 10 -num_alignments 10
#BLAST	00:00:00	Deleted BLAST results file: /srv/slimsuite/scratch/prod/slimfinder/24031400014/slimfinder.self.blast
#CLUST	00:00:00	10 clusters generated from 10 keys (maxdis=1.00)
#DICT	00:00:00	Made "short" seqName dictionary: 10 seq; 10 keys.

Generated Motif log part 2

3.2.6 Run Information

The run information can be found here, these will be valid for 1 month:
Run Info of hm03, Human genome
Run Info of yst08r, Yeast

4 Result :

4.1 Experimental Configuration :

The detailed result can be viewed at the github repository [link in reference].

4.1.1 Randomized Motif Search :

- **Number of Sequences :** 10 for **hm03** and 11 for **yst09r**
- **Sequence Length :** Each sequence is 1500 nucleotides long for **hm03** and 1000 nucleotides for **yst08r**.
- **K-mer Lengths :** The algorithm was run with k-mer lengths of 5, 10, and 15.
- **Maximum Iterations :** Set to 1000.
- **Restart Threshold :** The algorithm employs a random restart after 10 iterations without improvement to avoid local maxima.

4.1.2 Gibbs Sampling :

- **Number of Sequences :** 10 for **hm03** and 11 for **yst08r**
- **Sequence Length :** Similar to Randomized Motif Search, 1500 nucleotides for **hm03** and 1000 for **yst08r**.
- **K-mer Length :** The output specifies a k-mer length of 10.
- **Number of Iterations :** The algorithm was run for 10,000 iterations, suggesting a thorough exploration of the motif space

4.1.3 rGADEM :

- **Working Directory:** Specified as `"D:/rgadem"`.
- **FASTA File Name:** `"result.fasta"`, indicating the source of sequences.
- **Sequence Processing:** Sequences are read from the FASTA file using the `readDNAStringSet` function from the Bioconductor `biostrings` package, indicating reliance on R's bioinformatics capabilities.
- **GADEM Parameters (inferred from sample output):**
  - **Number of Sequences:** 10 for one run with sequence lengths averaging to 1500. For another run, 11 sequences with an average length of 1000.
  - **Number of GA Generations:** 5, setting the depth of the genetic algorithm's evolutionary search.
  - **Population Size:** 100, defining the number of motif candidates considered in each generation.
  - **PWM Score P-Value Cutoff:** 2×10^{-4}, used for declaring a binding site.
  - **E-Value Cutoff:** 0, for motif declaration, suggesting that all motifs meeting the PWM score cutoff are considered.
  - **Number of EM Steps:** 40, indicating the extent of refinement for each motif candidate.
  - **Minimal Number of Sites for a Motif:** 2, setting the minimum occurrences for considering a pattern as a motif.

4.1.4 SLiMFinder :

- **Sequence Length :** All sequences analyzed were 1000 nucleotides long.
- **Motif Significance :** All motifs identified had a significance score of 1.00, indicating high confidence in their relevance.
- **Motif Ranking :** Motifs were ranked to highlight the most significant and recurring patterns across the sequences.
- **Motif Positioning :** The exact start and end positions of each motif within the sequences were meticulously reported, allowing for precise mapping of motif locations.

4.2 Comparison :

To conduct a thorough comparison of the four methods for motif discovery, we analyze key aspects such as motif quality, time complexity, and other relevant metrics. The comparison is based on the provided outputs for each method.

4.2.1 Randomized Motif Discovery Algorithm:

- **Motif Quality :** Utilizes entropy to assess motif consensus. Lower entropy values indicate clearer consensus among identified motifs. For instance, 10-mer motifs in the hm03 dataset showed an entropy of 1.3103, suggesting a significant motif consensus.
- **Computational Efficiency :** The algorithm employs a restart mechanism to escape local maxima, enhancing the search efficiency. The maximum iterations are set to 1000, with a restart threshold at 10, balancing exploration and computational demand.

4.2.2 Gibbs Sampling Algorithm :

- **Motif Quality :** Similar to the Randomized Motif Discovery Algorithm, it uses entropy to measure motif consensus. A notable observation is the entropy of 1.9307 for 10-mer motifs in the hm03 dataset, indicating slightly less consensus compared to the Randomized method.
- **Iterations and Exploration :** The algorithm allows for 10,000 iterations, providing a more exhaustive search at the expense of higher computational time.

4.2.3 rGADEM :

- **Unique Motifs and Exploration Depth :** rGADEM is capable of identifying multiple unique motifs within a single run, showcasing its extensive exploration capabilities. The tool's configuration, with 5 GA generations and 40 EM steps, directly impacts the refinement of identified motifs.
- **Motif Representation :** The tool's output provides insights into the genetic algorithm's performance, including the fitness scores of motifs, which help gauge the relevance and quality of identified motifs.

4.2.4 SLiMFinder :

- **Diversity and Significance :** SLiMFinder outputs a diverse range of motifs, each with a significance score. This approach not only uncovers a broad spectrum of potential motifs but also ranks them based on their significance, offering a detailed motif landscape
- **Pattern Complexity :** The tool is adept at identifying complex motifs, as evidenced by the variety of patterns and ranks in its output, which can be particularly useful for in-depth analyses of motif complexity within biological sequences

4.3 Comparative Insights :

- **Entropy and Motif Consensus :** The Randomized Motif Discovery Algorithm tends to identify motifs with stronger consensus (lower entropy) compared to the Gibbs Sampling, suggesting its efficiency in finding clear motifs within datasets.
- **Computational Time :** Gibbs Sampling's higher iteration count implies a more thorough but time-consuming search process, whereas the Randomized Algorithm's restart mechanism optimizes the search, potentially reducing computational time.
- **Exploration and Diversity :** rGADEM demonstrates deep exploration capabilities by identifying multiple unique motifs, making it suitable for complex datasets with varied motif landscapes. SLiMFinder's extensive output, including diverse motifs and significance scores, offers a comprehensive motif analysis, useful for detailed investigations.

5 Conclusion :

Each motif discovery method presents unique strengths: The Randomized Motif Discovery Algorithm is efficient for identifying strong motifs, Gibbs Sampling offers thorough exploration at the cost of higher computational time, rGADEM excels in deep motif exploration, and SLiMFinder provides a comprehensive analysis with detailed motif rankings. The choice among these methods should be informed by the specific research objectives, dataset characteristics, computational resource availability, and the desired depth of motif analysis.

6 Reference

1. Experiment Codes
   GitHub Repository: Excellence, R. (*Year*). CSE-463-Bioinformatics-Assignment. GitHub. `https://github.com/excellencior/CSE-463-Bioinformatics-Assignment`

2. Human Genome Dataset
   Bioconductor. (*Year*). BSgenome.Hsapiens.UCSC.hg38. Retrieved from `https://bioconductor.org/packages/release/data/annotation/html/BSgenome.Hsapiens.UCSC.hg38.html`

3. rGADEM Tutorial
   Bioconductor. (*Year*). rGADEM: de novo motif discovery. Retrieved from `https://bioconductor.org/packages/release/bioc/vignettes/rGADEM/inst/doc/rGADEM.pdf`

4. SLimFinder Tutorial
   SLiMSuite. (*Year*). SLimFinder. Retrieved from `http://www.slimsuite.unsw.edu.au/servers/slimfinder.php`

5. Experiment's Result
   GitHub Repository: Excellence, R. (*Year*). Experiment's Result. GitHub. `https://github.com/excellencior/CSE-463-Bioinformatics-Assignment/blob/main/Code/saved_results.txt`