Bioinformatics Assignment
Motif Finding

CSE 463 : Introduction to Bioinformatics

Supervisor: Dr. Muhammad Ali Nayeem

Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology

Group Members

Arnab Bhattacharjee — 1905065
Abir Muhtasim — 1905066
Salman Sayeed — 1905079
Md. Raihan Sobhan — 1905095
Md. Labid Al Nahiyan — 1905110

March 14, 2024

Contents

Chapter 1

Methods

1.1 Randomized Motif Search

In randomized motif search, we first start with randomly selected k-mers in each string from DNA. Then we create a profile from those random k-mers. We keep on improving the motifs using the profile and improve the profile using the new motif. We run this algorithm many times at random starting positions and hope that we will get close to the optimum solution.

We have implemented randomized motif search in our code. The code is in the snipped below :

```
    #include<bits/stdc++.h>
using namespace std;

string find_random_motif(string str, int k, int n)
{
    int pos = rand() % (n - k);
    return str.substr(pos, k);
}

vector<string> find_profile_most_motifs(int t, int n, int k,
↪  string dna[], double profile[][100])
{
    double score, best_score;
    string temp;
    vector<string> profile_most_motifs;

    for(int i = 0; i < t; ++i)
    {
        best_score = 0;
        for(int j = 0; j < n-k; ++j)
        {
            score = 1;
            for(int pos = 0; pos < k; ++pos)
            {
                switch(dna[i][j+pos])
                {
                    case 'A':
                    case 'a': score *= profile[0][pos];
                            break;
                    case 'C':
                    case 'c': score *= profile[1][pos];
                            break;
                    case 'G':
                    case 'g': score *= profile[2][pos];
                            break;
                    case 'T':
                    case 't': score *= profile[3][pos];
                            break;
                }
            }
            if(score > best_score)
            {
                best_score = score;
                temp = dna[i].substr(j, k);
            }
        }
        profile_most_motifs.push_back(temp);
    }

    return profile_most_motifs;
}

void random_motif_search(int n, int t, int k, int it, string
 ↪  dna[])
{
    double profile[5][100];
    vector<string> profile_most_motifs, random_motifs;

    for(int i = 0; i < it; ++i)
    {
        profile_most_motifs.clear();
        random_motifs.clear();

        for(int j = 0; j < t; ++j)
            random_motifs.push_back( find_random_motif(dna[j], k,
             ↪  n) );
        int aa, cc, gg, tt;
        for(int j = 0; j < k; ++j)
        {
            aa = cc = gg = tt = 0;
            for(int l = 0; l < t; ++l)
            {
                switch(random_motifs[l][j])
                {
                    case 'A':
                    case 'a': ++aa;
                            break;
                    case 'C':
                    case 'c': ++cc;
                            break;
                    case 'G':
                    case 'g': ++gg;
                            break;
                    case 'T':
                    case 't': ++tt;
                            break;
                }
            }
            profile[0][j] = aa / (double) t;
            profile[1][j] = cc / (double) t;
            profile[2][j] = gg / (double) t;
            profile[3][j] = tt / (double) t;
        }

        profile_most_motifs = find_profile_most_motifs(t, n, k,
         ↪  dna, profile);
    }

    cout << "Random Motifs:" << endl;
    for(int i = 0; i < t; ++i)
        cout << random_motifs[i] << endl;
    cout << endl;

    char nuces[] = {'A', 'C', 'G', 'T'};
    cout << "Profile:" << endl;
    for(int i = 0; i < 4; ++i)
    {
        cout << nuces[i] << ": ";
        for(int j = 0; j < k; ++j)
            cout << fixed << setprecision(4) << profile[i][j] <<
             ↪  " ";
        cout << endl;
    }
    cout << endl;

    cout << "Profile Most Motifs:" << endl;
    for(int i = 0; i < t; ++i)
        cout << profile_most_motifs[i] << endl;

    return;
}

int main()
{
    srand(time(NULL));
    int n, t, k, it;
    string dna[100];
    freopen("RMS_yst08r.txt","r",stdin);
    cin >> n >> t >> k >> it;
    for(int i = 0; i < t; ++i)
        cin >> dna[i];

    random_motif_search(n, t, k, it, dna);

    return 0;
}
```

1.2 Gibbs Sampling

In Gibbs sampling for motif finding, the algorithm iteratively samples potential motif positions within each sequence while keeping the positions of the motifs in the other sequences fixed. It proceeds more slowly and chooses new l-mers at random increasing the odds that it will converge to the correct solution.

Working Procedure:

1. Randomly choose starting positions $s = (s_1, \ldots, s_t)$ and form the set of l-mers associated with these starting positions.
2. Randomly choose one of the t sequences.
3. Create a profile P from the other $t - 1$ sequences.
4. For each position in the removed sequence, calculate the probability that the l-mer starting at that position was generated by P.
5. Choose a new starting position for the removed sequence at random based on the probabilities calculated in step 4.

6. Repeat steps 2-5 until there is no improvement.

We have implemented randomized motif search in our code. The code is in the snipped below :

```
import random
from datetime import datetime

def symbolToNumber(symbol):
        if symbol == "A":
                return 0
        if symbol == "C":
                return 1
        if symbol == "G":
                return 2
        if symbol == "T":
                return 3

def numberToSymbol(x):
        if x == 0:
                return "A"
        if x == 1:
                return "C"
        if x == 2:
                return "G"
        if x == 3:
                return "T"

def profileRandom(k, profile, text):
    probs = []
    for i in range(0,len(text) - k +1):
        prob = 1.0
        pattern = text[i:i+k]
        for j in range(k):
            l = symbolToNumber(pattern[j])
            prob *= profile[l][j]
        probs.append(prob)
    r = myRandom(probs)
    return r

def profileForm(motifs):
        k = len(motifs[0])
        profile = [[1 for i in range(k)] for j in range(4)]
        for x in motifs:
                for i in range(len(x)):
                        j = symbolToNumber(x[i])
                        profile[j][i] += 1
        for x in profile:
                for i in range(len(x)):
                        x[i] = x[i]/len(motifs)
        return profile

def consensus(profile):
        str = ""
        for i in range(len(profile[0])):
                max = 0
                loc = 0
                for j in range(4):
                        if profile[j][i] > max:
                                loc = j
                                max = profile[j][i]
                str+=numberToSymbol(loc)
        return str

def score(motifs):
        profile = profileForm(motifs)
        cons = consensus(profile)
        score = 0
        for x in motifs:
                for i in range(len(x)):
                        if cons[i] != x[i]:
                                score += 1
        return score

def myRandom(dist):
    s = 0.0
    for x in dist:
        s+= x
    i = random.random()
    partial = 0.0
    for x in range(len(dist)):
        partial += dist[x]
        if partial/s >= i:
            return x

def gibbsSampler(dna, k, t, n):
    bestMotifs = []
    motifs = []
    for x in range(t):
        i = random.randint(0, len(dna[x])-k)
        motifs.append(dna[x][i:i+k])
    bestMotifs = motifs[:]
    for i in range(n):
        j = random.randint(0,t-1)
        profile = profileForm(motifs[:j] + motifs[j+1:])
        r = profileRandom(k, profile, dna[j])
        motifs[j] = dna[j][r:r+k]
        if score(motifs) < score(bestMotifs):
            bestMotifs = motifs[:]
    return bestMotifs

start = datetime.now()
with open("hm03.txt") as file:

    k, t, n = [int(x) for x in file.readline().split()]
    dna = []
    for line in file:
        dna.append(line.rstrip())

best = gibbsSampler(dna, k, t, n)
s = score(best)

for x in range(20):
    sample = gibbsSampler(dna, k, t, n)
    if score(sample) < s:
        s = score(sample)
        best = sample[:]
for b in best:
        print(b)


end = datetime.now()
print("Runtime : ",end-start)
```

Chapter 2

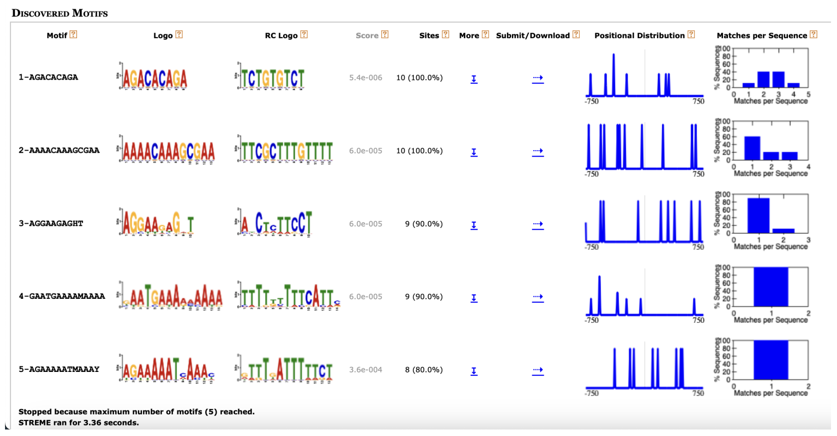
Software

We ran all the files in both STREME and MEME$_{C}hIPtoolsandtheoutputwasasfollows$:

2.1 STREME

For this tool we just upload the DNA file that we want to find the motif in. But we need to provide the file in a different format. We need to have IDs before each line. After this, we simply upload our dna and an output file would be generated. We can view the HTML code and view this. Some of the outputs are as follows:

2.1.1 For file hm03.txt



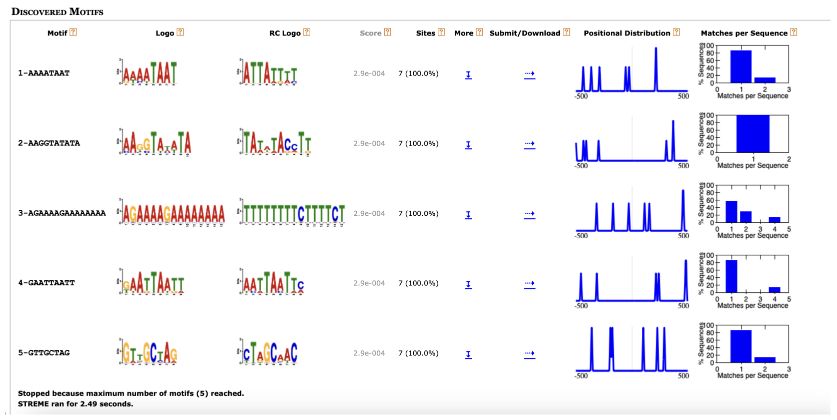
DISCOVERED MOTIFS

Motif	Logo	RC Logo	Score	Sites	More	Submit/Download	Positional Distribution	Matches per Sequence
1-AGACACAGA			5.4e-006	10 (100.0%)				
2-AAAACAAAGCGAA			6.0e-005	10 (100.0%)				
3-AGGAAGAGHT			6.0e-005	9 (90.0%)				
4-GAATGAAAAMAAAA			6.0e-005	9 (90.0%)				
5-AGAAAAATMAAAY			3.6e-004	8 (80.0%)				

Stopped because maximum number of motifs (5) reached.
STREME ran for 3.36 seconds.

Figure 2.1: Output for the file hm03.txt

2.1.2 For file yst04r.txt



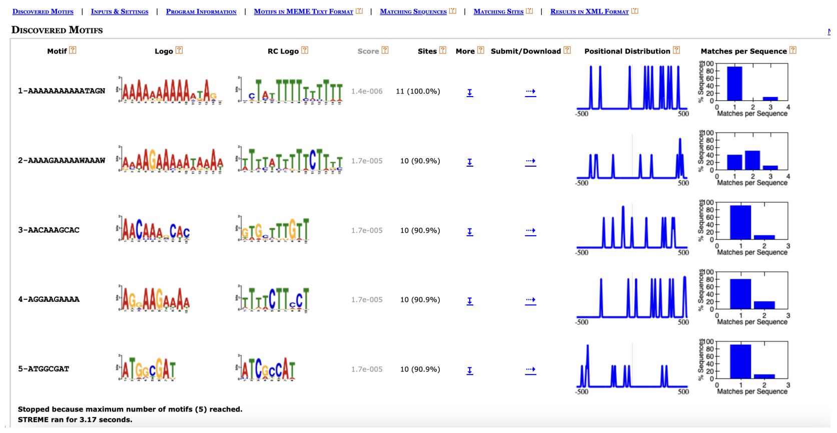
Discovered Motifs

Motif	Logo	RC Logo	Score	Sites	More	Submit/Download	Positional Distribution	Matches per Sequence
1-AAAATAAT			2.9e-004	7 (100.0%)				
2-AAGGTATATA			2.9e-004	7 (100.0%)				
3-AGAAAAGAAAAAAAA			2.9e-004	7 (100.0%)				
4-GAATTAATT			2.9e-004	7 (100.0%)				
5-GTTGCTAG			2.9e-004	7 (100.0%)				

Stopped because maximum number of motifs (5) reached.
STREME ran for 2.49 seconds.

Figure 2.2: Output for the file yst04r.txt

2.1.3 For file yst08r.txt



Discovered Motifs | Inputs & Settings | Program Information | Motifs in MEME Text Format | Matching Sequences | Matching Sites | Results in XML Format

Discovered Motifs

Motif	Logo	RC Logo	Score	Sites	More	Submit/Download	Positional Distribution	Matches per Sequence
1-AAAAAAAAAAAATAGN			1.4e-006	11 (100.0%)				
2-AAAAGAAAAAWAAAW			1.7e-005	10 (90.9%)				
3-AACAAAGCAC			1.7e-005	10 (90.9%)				
4-AGGAAGAAAA			1.7e-005	10 (90.9%)				
5-ATGGCGAT			1.7e-005	10 (90.9%)				

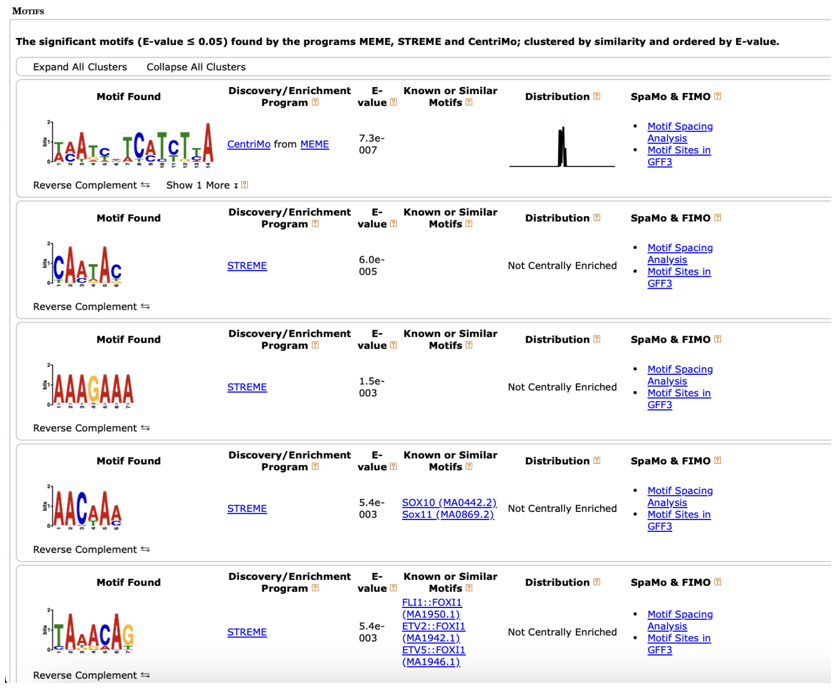
Stopped because maximum number of motifs (5) reached.
STREME ran for 3.17 seconds.

Figure 2.3: Output for the file yst08r.txt

2.2 MEME-ChIP

For this tool we just upload the DNA file that we want to find the motif in. But we need to provide the file in a different format. We need to have IDs before each line. After this, we simply upload our dna and an output file would be generated. We can view the HTML code and view this. Some of the outputs are as follows:

2.2.1 For file hm03.txt



Motifs

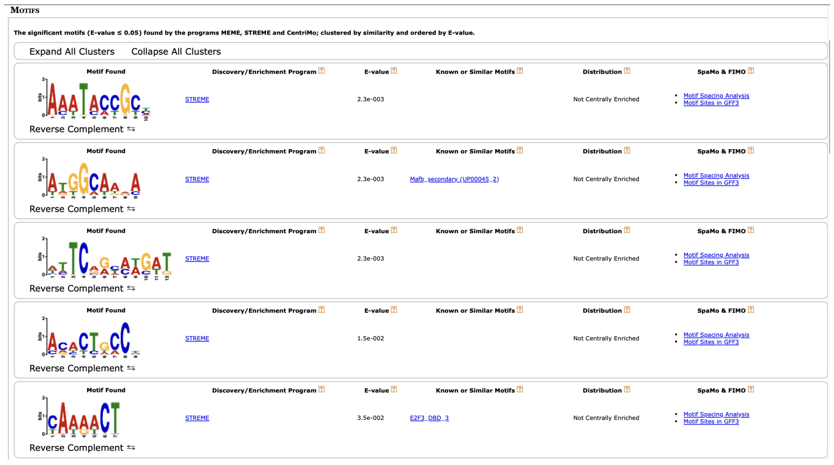
The significant motifs (E-value ≤ 0.05) found by the programs MEME, STREME and CentriMo; clustered by similarity and ordered by E-value.

Expand All Clusters   Collapse All Clusters

Motif Found	Discovery/Enrichment Program	E-value	Known or Similar Motifs	Distribution	SpaMo & FIMO
Reverse Complement ⇆ Show 1 More	CentriMo from MEME	7.3e-007			• Motif Spacing Analysis • Motif Sites in GFF3
Reverse Complement ⇆	STREME	6.0e-005		Not Centrally Enriched	• Motif Spacing Analysis • Motif Sites in GFF3
Reverse Complement ⇆	STREME	1.5e-003		Not Centrally Enriched	• Motif Spacing Analysis • Motif Sites in GFF3
Reverse Complement ⇆	STREME	5.4e-003	SOX10 (MA0442.2) Sox11 (MA0869.2)	Not Centrally Enriched	• Motif Spacing Analysis • Motif Sites in GFF3
Reverse Complement ⇆	STREME	5.4e-003	FLI1::FOXI1 (MA1950.1) ETV2::FOXI1 (MA1942.1) ETV5::FOXI1 (MA1946.1)	Not Centrally Enriched	• Motif Spacing Analysis • Motif Sites in GFF3

Figure 2.4: Output for the file hm03.txt

2.2.2 For file yst04r.txt



Motifs

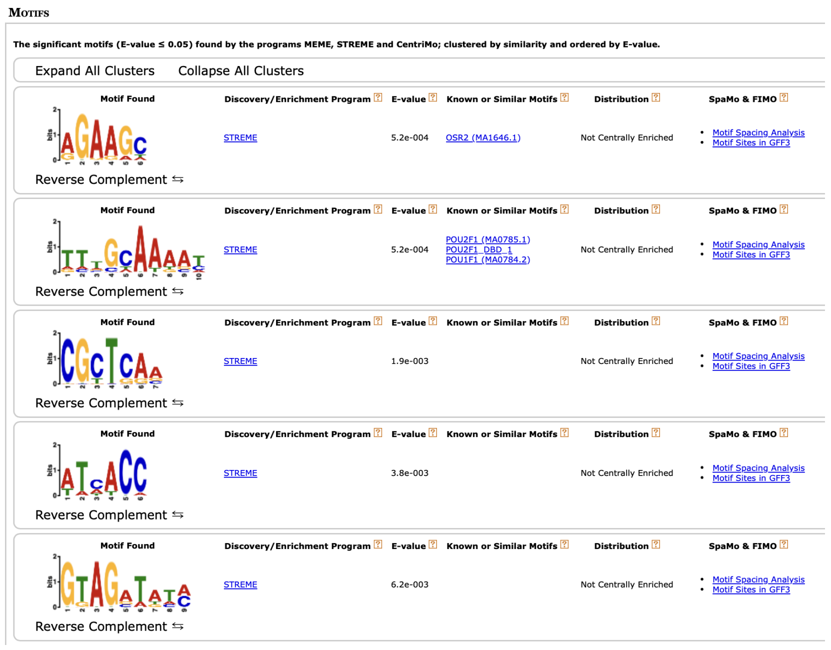
The significant motifs (E-value ≤ 0.05) found by the programs MEME, STREME and CentriMo; clustered by similarity and ordered by E-value.

Expand All Clusters   Collapse All Clusters

Motif Found	Discovery/Enrichment Program	E-value	Known or Similar Motifs	Distribution	SpaMo & FIMO
Reverse Complement ⇆	STREME	2.3e-003		Not Centrally Enriched	• Motif Spacing Analysis • Motif Sites in GFF3
Reverse Complement ⇆	STREME	2.3e-003	Mafb_secondary (UP00045_2)	Not Centrally Enriched	• Motif Spacing Analysis • Motif Sites in GFF3
Reverse Complement ⇆	STREME	2.3e-003		Not Centrally Enriched	• Motif Spacing Analysis • Motif Sites in GFF3
Reverse Complement ⇆	STREME	1.5e-002		Not Centrally Enriched	• Motif Spacing Analysis • Motif Sites in GFF3
Reverse Complement ⇆	STREME	3.5e-002	E2F3_DBD_3	Not Centrally Enriched	• Motif Spacing Analysis • Motif Sites in GFF3

Figure 2.5: Output for the file yst04r.txt

2.2.3 For file yst08r.txt



Motifs

The significant motifs (E-value ≤ 0.05) found by the programs MEME, STREME and CentriMo; clustered by similarity and ordered by E-value.

Expand All Clusters   Collapse All Clusters

Motif Found	Discovery/Enrichment Program	E-value	Known or Similar Motifs	Distribution	SpaMo & FIMO
Reverse Complement ⇆	STREME	5.2e-004	OSR2 (MA1646.1)	Not Centrally Enriched	• Motif Spacing Analysis • Motif Sites in GFF3
Reverse Complement ⇆	STREME	5.2e-004	POU2F1 (MA0785.1) POU2F1_DBD_1 POU1F1 (MA0784.2)	Not Centrally Enriched	• Motif Spacing Analysis • Motif Sites in GFF3
Reverse Complement ⇆	STREME	1.9e-003		Not Centrally Enriched	• Motif Spacing Analysis • Motif Sites in GFF3
Reverse Complement ⇆	STREME	3.8e-003		Not Centrally Enriched	• Motif Spacing Analysis • Motif Sites in GFF3
Reverse Complement ⇆	STREME	6.2e-003		Not Centrally Enriched	• Motif Spacing Analysis • Motif Sites in GFF3

Figure 2.6: Output for the file yst08r.txt

2.3 Commands to run

2.4 STREME

2.4.1 For file hm03.txt

```
streme –verbosity 1 –oc . –dna –totallength 4000000 –time 14400 –minw 8
–maxw 15 –thresh 0.05 –align center –p hm03.txt
```

2.4.2 For file yst04r.txt

```
streme –verbosity 1 –oc . –dna –totallength 4000000 –time 14400 –minw 8
–maxw 15 –thresh 0.05 –align center –p yst04r.txt
```

2.4.3 For file yst08r.txt

```
streme –verbosity 1 –oc . –dna –totallength 4000000 –time 14400 –minw 8
–maxw 15 –thresh 0.05 –align center –p yst08r.txt
```

2.5 MEME-ChIP

2.5.1 For file hm03.txt

meme-chip -oc . -time 240 -ccut 100 -dna -order 2 -minw 6 -maxw 15 -db
db/motif$_{d}atabases/MOUSE/uniprobe_{m}ouse.meme-dbdb/motif_{d}atabases/JASPAR/JASPAR2022_{C}ORE_{v}$
$redundant_{v}2.meme-dbdb/motif_{d}atabases/EUKARYOTE/jolma2013.meme-$
$meme-modzoops-meme-nmotifs3-meme-searchsize100000-streme-$
$pvt0.05-streme-aligncenter-streme-totallength4000000-centrimo-$
$score5.0-centrimo-ethresh10.0hm03.txt$

2.5.2 For file yst04r.txt

meme-chip -oc . -time 240 -ccut 100 -dna -order 2 -minw 6 -maxw 15 -db
db/motif$_{d}atabases/MOUSE/uniprobe_{m}ouse.meme-dbdb/motif_{d}atabases/JASPAR/JASPAR2022_{C}ORE_{v}$
$redundant_{v}2.meme-dbdb/motif_{d}atabases/EUKARYOTE/jolma2013.meme-$
$meme-modzoops-meme-nmotifs3-meme-searchsize100000-streme-$
$pvt0.05-streme-aligncenter-streme-totallength4000000-centrimo-$
$score5.0-centrimo-ethresh10.0yst04r.txt$

2.5.3 For file yst08r.txt

meme-chip -oc . -time 240 -ccut 100 -dna -order 2 -minw 6 -maxw 15 -db
db/motif$_{d}atabases/MOUSE/uniprobe_{m}ouse.meme-dbdb/motif_{d}atabases/JASPAR/JASPAR2022_{C}ORE_{v}$
$redundant_{v}2.meme-dbdb/motif_{d}atabases/EUKARYOTE/jolma2013.meme-$
$meme-modzoops-meme-nmotifs3-meme-searchsize100000-streme-$

$pvt0.05 - streme - aligncenter - streme - totallength4000000 - centrimo - score5.0 - centrimo - ethresh10.0yst08r.txt$

2.6 Scripts to run

Chapter 3

Results

3.1 Experiment Configuration

The tools provided us with motifs with size ranging from 8 to 15. We ran our algo for three of the input files for each of the values 8,10,12,15.

3.2 Comparison

3.3 Randomized Motif Search

We wrote a code following the pseudocode of the randomized motif search algorithm. Now we ran this code several times each time with a different motif length ranging from 8 to 15. These are some of the values that we received

3.3.1 For file hm03.txt

When k=15, the output motifs were :

```
    Random Motifs: ATCGGGTG
GTTTCTCA
TTGGGTGT

TTTTCTTT
CTTCCGGT
GGGACTGA
CTTCACTT
ATTGTGGA
AAGGCGAG
CGGGGCAG
```

```
Profile:
A: 0.3000 0.1000 0.0000 0.1000 0.1000 0.0000 0.2000 0.3000
C: 0.3000 0.0000 0.1000 0.2000 0.5000 0.2000 0.1000 0.0000
G: 0.2000 0.2000 0.4000 0.5000 0.3000 0.4000 0.4000 0.3000
T: 0.2000 0.7000 0.5000 0.2000 0.1000 0.4000 0.3000 0.4000

Profile Most Motifs:
CTGGCTGT
ATTGCTGG
TTTGCTGT
ATGGCTTT
CTGGCTGA
CTGGGTGA
CTTGCGGT
ATTGCTGT
CTGGCTTT
```

When k=10 the output was :

```
Random Motifs:
TTATTCACTA
GTTGCAGCTT
AGAGCTGCTA
AAGATTATAG
CTCCCACTCG
AGAATGAGGC
TCCAAAAGTA
CCTGATGAAG
TTAAGGGAGT
GAAAATGAGG

Profile:
A: 0.3000 0.2000 0.5000 0.5000 0.3000 0.3000 0.4000 0.3000 0.2000 0.3000
C: 0.2000 0.2000 0.2000 0.1000 0.3000 0.1000 0.1000 0.3000 0.1000 0.1000
G: 0.2000 0.2000 0.1000 0.3000 0.1000 0.2000 0.5000 0.2000 0.3000 0.4000
T: 0.3000 0.4000 0.2000 0.1000 0.3000 0.4000 0.0000 0.2000 0.4000 0.2000

Profile Most Motifs:
ATAATAATTG
ATAATTAAAG
ATAAAAGCTG
TTAACTGAAG
ATAAATGGGG
TTAACTGAAG
AAAATTACTG
AAAATTGCTG
AGAATGAATG
```

```
GAAAATGAGG
```

When k=12 the output was :

```
Random Motifs:
CACTAAACAGTCCTT
TCATGAATAATAATT
GCTTAAAAACCTAGA
ACAGAGAATTTGGGG
GACTGGTCACTCACC
TACTCACCCCATGGA
ACATACAATGCAGTA
TCTCATTATGCCTAC
CCCAGGTGCCCGCCC
CATCCTCCCCCAAAA

Profile:
A: 0.2000 0.4000 0.3000 0.1000 0.5000 0.4000 0.5000 0.4000 0.4000 0.1000 0.1000
0.3000 0.4000 0.2000 0.4000
C: 0.3000 0.6000 0.4000 0.2000 0.2000 0.1000 0.2000 0.4000 0.3000 0.5000 0.5000
0.3000 0.2000 0.2000 0.3000
G: 0.2000 0.0000 0.0000 0.1000 0.3000 0.3000 0.0000 0.1000 0.0000 0.3000 0.0000
0.2000 0.3000 0.3000 0.1000
T: 0.3000 0.0000 0.3000 0.6000 0.0000 0.2000 0.3000 0.1000 0.3000 0.1000 0.4000
0.2000 0.1000 0.3000 0.2000

    Profile Most Motifs:
CACTAAACAGTCATT
CACTATAAACTGTGA
GCTTAAAAACCTAGA
TACTAAACACTGAGA
GACTGGTCACTCACC
CCCCAAAAAGTAGGA
GACTAGTACCTGGGA
AACTGGTCACCCAGA
GCATGAACAGTGGGT
TCCTGGAACCTTCTT
```

When k=15 the output was :

```
Random Motifs:
CACTGGACCCAGCTT
GTGCCTGACACAGGG
TCATGGCAAAGGTCT
TAGTTAATTAGTACC
TGTAACAATGAGCAC
ACCCGTAGTCCCGGC
TTTTATGCTGTATAA
```

GAGAAACAGGACTGT
TCAACAGGAATCCTC
CAATTCTGACAACAC

Profile:
A: 0.1000 0.4000 0.3000 0.3000 0.3000 0.3000 0.4000 0.4000 0.3000 0.4000 0.4000 0.3000 0.1000 0.3000 0.1000
C: 0.2000 0.3000 0.2000 0.2000 0.2000 0.2000 0.2000 0.2000 0.2000 0.3000 0.2000 0.3000 0.4000 0.2000 0.5000
G: 0.2000 0.1000 0.3000 0.0000 0.3000 0.2000 0.3000 0.3000 0.1000 0.3000 0.2000 0.3000 0.2000 0.3000 0.1000
T: 0.5000 0.2000 0.2000 0.5000 0.2000 0.3000 0.1000 0.1000 0.4000 0.0000 0.2000 0.1000 0.3000 0.2000 0.3000

Profile Most Motifs:
TAATCTAATGGGTGT
TAGAGAAATATGTCC
TCAACTAATAAGTCC
TAGTAAAAAAGACAC
TCATCTGAAGAAGGC
TCGACTGATAAACGC
GAGTATAGTCTGTGT
TCATTCAACAAATAT
CACCAAAATAAACAT
GCATATAGAAGGCAC

3.3.2 For file yst04r.txt

When k=8, the output motifs were :
Random Motifs:
ACAGATCA
AATACATT
CCACACCA
ATAAAAAA
AAACACTA
CACGCCTT
CCTGGTCA

Profile:
A: 0.5714 0.4286 0.5714 0.2857 0.5714 0.2857 0.1429 0.7143
C: 0.4286 0.4286 0.1429 0.2857 0.2857 0.4286 0.4286 0.0000
G: 0.0000 0.0000 0.0000 0.4286 0.1429 0.0000 0.0000 0.0000
T: 0.0000 0.1429 0.2857 0.0000 0.0000 0.2857 0.4286 0.2857

Profile Most Motifs:

AAAGAACA
AAAGATTA
AAAAACTA
CAAGACTA
AAAAACCA
AAACACTA
AAAGATTA
When k=10, the output motifs were :
Random Motifs:
TGGTCTTTTC
TGAGGACCTT
AAAAACACCA
GTTACAATAT
AGAAAAATAC
TTTGACATTG
TAAAATTGAT

Profile:
A: 0.2857 0.2857 0.5714 0.5714 0.5714 0.4286 0.5714 0.0000 0.4286 0.1429
C: 0.0000 0.0000 0.0000 0.0000 0.2857 0.2857 0.1429 0.2857 0.1429 0.2857
G: 0.1429 0.4286 0.1429 0.2857 0.1429 0.0000 0.0000 0.1429 0.0000 0.1429
T: 0.5714 0.2857 0.2857 0.1429 0.0000 0.2857 0.2857 0.5714 0.4286 0.4286

Profile Most Motifs:
TGAAACTTAC
TAAGACTTTT
TATAAAATTC
TGTACAATAT
AGAAAAATTT
TGAAAAATTC
AAAACAATAT
When k=12, the output motifs were :
Random Motifs:
TTAATTTTTTTTCT
AAGAGAGAAGACCCA
CTGTCTCGAGGGTGC
TGTAGCCCTAGACTT
AGAAAGCATACTATA
CAAGTGTCGACGCTG
AAATGTGTTCCGCAC

Profile: A: 0.4286 0.4286 0.5714 0.5714 0.1429 0.1429 0.0000 0.2857 0.2857 0.4286 0.1429 0.1429 0.1429 0.1429 0.2857
C: 0.2857 0.0000 0.0000 0.0000 0.1429 0.1429 0.4286 0.2857 0.0000 0.1429 0.4286 0.1429 0.5714 0.2857 0.2857
G: 0.0000 0.2857 0.2857 0.1429 0.4286 0.2857 0.2857 0.1429 0.1429 0.2857 0.2857

0.4286 0.0000 0.1429 0.1429
T: 0.2857 0.2857 0.1429 0.2857 0.2857 0.4286 0.2857 0.2857 0.5714 0.1429 0.1429 0.2857 0.2857 0.4286 0.2857

Profile Most Motifs:
CTAATTCGTAGTTTT
AATAATCCTACGTTA
TAATGTCTTACTTCT
TAAAGTCCAATGCTA
AGAAAGCATACTATA
CTAATTGATACATCT
CAAAGTTATTCTCTA
When k=15, the output motifs were :
Random Motifs:
AAGTAATTATCTACT
AAATCTATAACTACA
GATCAATCAGTACTA
CTAAAGGAAAAAATT
TTATCTACTGATCCG
AAACGAAGGCTATCG
TATTATTCTCTCTTG

Profile:
A: 0.4286 0.7143 0.5714 0.1429 0.5714 0.4286 0.4286 0.1429 0.5714 0.2857 0.2857 0.4286 0.4286 0.0000 0.2857
C: 0.1429 0.0000 0.0000 0.2857 0.2857 0.0000 0.0000 0.4286 0.0000 0.2857 0.2857 0.1429 0.2857 0.5714 0.0000
G: 0.1429 0.0000 0.1429 0.0000 0.1429 0.1429 0.1429 0.1429 0.1429 0.2857 0.0000 0.0000 0.0000 0.0000 0.4286
T: 0.2857 0.2857 0.2857 0.5714 0.0000 0.4286 0.4286 0.2857 0.2857 0.1429 0.4286 0.4286 0.2857 0.4286 0.2857

Profile Most Motifs:
AAGTAATTATCTACT
AAATCTATAACTACA
TTATAAACATCAACA
AATTAAAGTCCAATG
TATTAATCAAATTTA
AAATATTTGACATTG
AAATAAACAATCACA

3.3.3 For file yst08r.txt

When k=8, the output motifs were :
Random Motifs:

```
TTCCACTA
TTGGTTAC
TCTGTTAG
ATCCTTTG
TCCTCGTT
GTTCTACT
ATGCTAAA
ACGAGAAA
CGCTCAAA
ATACATAT
AGTTATAA

Profile:
A: 0.4545 0.0000 0.0909 0.0909 0.2727 0.3636 0.6364 0.4545
C: 0.0909 0.2727 0.3636 0.4545 0.1818 0.0909 0.0909 0.0909
G: 0.0909 0.1818 0.2727 0.1818 0.0909 0.0909 0.0000 0.1818
T: 0.3636 0.5455 0.2727 0.2727 0.4545 0.4545 0.2727 0.2727

Profile Most Motifs:
TTTCATAA
ATCCAAAA
TTCCTTAA
ATTCTTAA
TTTTTTAA
TTCCAAAA
ATCTTTAA
ATGTTTAA
TTTCAAAA
ATCCTTTA
ATTCTAAA
```

When k=10, the output motifs were :

```
Random Motifs:
ATCGAAGCCG
TGGTTCATTA
CTCTCCTTGC
AATGGAAAAA
GGGCGACGAA
AGCTACAGTT
CTTTCGATGA
GTTCCTTGAG
CCGCCGACGA
AAAACCCAGA
AGGATGAGCC

   Profile:
A: 0.4545 0.1818 0.0909 0.1818 0.1818 0.2727 0.5455 0.1818 0.2727 0.5455
```

```
C: 0.2727 0.0909 0.2727 0.2727 0.4545 0.3636 0.1818 0.1818 0.1818 0.1818
G: 0.1818 0.3636 0.3636 0.1818 0.1818 0.2727 0.0909 0.3636 0.3636 0.1818
T: 0.0909 0.3636 0.2727 0.3636 0.1818 0.0909 0.1818 0.2727 0.1818 0.0909

Profile Most Motifs:
ATCTCCACGA
CGTTCCAAAA
ATGTGCAAGA
ATTTCCAAGA
ATGGCAACGA
CTGTCCAGGA
CTTTCGATGA
ATGACGAGAA
CTGGCAAGGA
AGGGCCAGAA
ATTTCCAAAA
```

When k=12, the output motifs were :

```
Random Motifs:
AGAGTGCATATC
ACAAACAATGGC
TGAAGGCCAAAT
AGTGATTTCCTT
ACGCTTGCTTTG
AGACGAGGTTTG
CATATTACATAT
TTGAGCTACCCT
TACACTGTAATG
ACTTCCCATACT
TTTGTATTCTTT

Profile:
A: 0.5455 0.1818 0.3636 0.4545 0.1818 0.1818 0.1818 0.3636 0.2727 0.3636 0.1818
0.0000
C: 0.0909 0.2727 0.0909 0.1818 0.1818 0.2727 0.2727 0.2727 0.2727 0.1818 0.1818
0.1818
G: 0.0000 0.3636 0.1818 0.2727 0.2727 0.1818 0.2727 0.0909 0.0000 0.0909 0.0909
0.2727
T: 0.3636 0.1818 0.3636 0.0909 0.3636 0.3636 0.2727 0.2727 0.4545 0.3636 0.5455
0.5455

Profile Most Motifs:
TCTAGCTATTTT
TTAATTTTTTTT
ACTATTTTTTTT
TGAATTATTTTT
ACAAGCTTATTT
```

AGTAACCTTTTT
AAAATTTTTTTT
TCTAGTGCTATT
TCAAGTGCATTT
TTAATTTTTTTT
AATATTTTTCTT
When k=15, the output motifs were :
Random Motifs:
GCCAAAAACTGATAT
CAACTAATACTATAA
ACACACCCCGCGTTT
GCAACGGCGGGATTC
CGAAAAAGCCAATCT
GACCAATGGCAGAGA
CTCCGCGCATCGCCG
TCCCCTGCCGGCTGT
TCAAAACGCTCAAAA
CAAGGGGGTGGTTTA
ATCGTTTACCAATGT

Profile:
A: 0.1818 0.2727 0.5455 0.3636 0.4545 0.4545 0.2727 0.1818 0.1818 0.0000 0.2727 0.5455 0.1818 0.2727 0.3636
C: 0.3636 0.4545 0.4545 0.4545 0.1818 0.1818 0.1818 0.3636 0.5455 0.3636 0.2727 0.0909 0.0909 0.1818 0.0909
G: 0.2727 0.0909 0.0000 0.1818 0.1818 0.1818 0.3636 0.3636 0.1818 0.3636 0.3636 0.2727 0.0000 0.2727 0.0909
T: 0.1818 0.1818 0.0000 0.0000 0.1818 0.1818 0.1818 0.0909 0.0909 0.2727 0.0909 0.0909 0.7273 0.2727 0.4545

Profile Most Motifs:
GCCAAAAACTGATAT
ACAAAAGCGGGATGA
ACACACCCCGCGTTT
AAAAATAGCTAATTT
CGAAAAAGCCAATCT
TTCCTAAGCCAATCT
CGAAAAAGGCAATAA
ACAAATTACCCATAA
AAAAAAAGCGCATGC
TAAAGATGCCGATTT
CAAAGGAGCCTATAT

3.4 Gibbs Sampler

We wrote a code following the pseudocode of the gibbs-sampler algorithm. Now we ran this code several times each time with a different motif length ranging from 8 to 15. These are some of the values that we received.

3.4.1 For file hm03.txt

When k=8, the output motifs were :

- AAAAAAAA
- AAAAAAAA
- AAAATAAA
- AAAAAAAA
- AAACAAAG
- AAAAAAAA
- AAGAACAA
- AAAATAAA
- AAAAAAAA
- ACACAAAA

The running time for this code was 2:02min
When k=10, the output motifs were :

- AAAAAAAATA
- AAACAAAATA
- AAACAAAATA
- AAAAGACACA
- AAAGCAAACA
- AAAAAAAACA
- ACAATAAACA
- AAATAAAACA
- AAAAAAAAAA
- ACAAAAATCA

The running time for this code was 2:25min
When k=12, the output motifs were :

- AAAAAAAATAAA
- AAACAAAATAAA
- AAACAAAATAAA
- AGAAAAAATAAT
- AAAAGAAGTAAC
- AAAAATAAAAAA
- AGACACAATAAA
- AAGTAAAATAAA
- AAAAAAAAAAAA
- AACAAAAATCAA

The running time for this code was 2:42min
When k=15, the output motifs were :

- AGCAAAAAAAATAAA
- AGCAAACAAAATAAA
- AGCAAACAAAATAAA
- GGAAAAAAATATGAA
- TGGAAAAGAAGTAAC
- ACAAAAATAAAAAAA
- AGAAGACACAATAAA
- AGTAAGTAAAATAAA
- AAAAAAAAAAAAAAA
- TGAACAAAAATCAAA

The running time for this code was 3:12min

3.4.2 For file yst04r.txt

When k=8, the output motifs were :

- TTTTTTTT
- TTTTTCTT
- TTTTTTTA
- TTTTTTTT
- TTTTGTTT
- TTTTTTTT
- TTTTTTTT

The running time for this code was 1:10min

When k=10, the output motifs were :

- TTTTTTTTCT
- ATTCTTTTCT
- TTTTTTTTAT
- TTTTTTTTCT
- TTATTTTACT
- TTTTTTTTCT
- TTTTTTTTTT

The running time for this code was 1:37min

When k=12, the output motifs were :

- TTTTTTTCTTTC
- TATTTTTCTTTT
- TATTTTTTTTAT
- TTTTTTTCTTTT
- TATTTTACTTCT
- TATTTTTCTTTT
- TATTTTTTTTTT

The running time for this code was 1:54min

When k=15, the output motifs were :

- TTCTTTTTCTCTTTT
- ATATTTTTCTTTTTA
- TTATTTTTATATTTA
- TTTTTTTTCTTTTCT
- TTATTTTACTTCTTT
- TTATTTTTCTTTTCT
- TTATTTTTTTTTTGT

The running time for this code was 2:16min

3.4.3 For file yst08r.txt

When k=8, the output motifs were :

- TCTTTTGT
- TTTTTTTT
- TTTTTTTT
- TTTTTTTT
- TTTTTTTT
- TATTTTTT
- TTTTTTTT
- TTTTTTTT
- TTTTTTCT
- TTTTTTTT
- TTCTTTTT

The running time for this code was 0:37min

3.4.4 For file yst08r.txt

When k=10, the output motifs were :

- TCTTTTGTTT
- TTTTTATTTT
- TTTTTTTTTT
- TTTTTTTTTT
- TTTTTTGTTT
- TTTTTATTTT
- TTTTTTTTTT
- TTTTTTTTTT
- TTTTTTCTAT
- TTTTTTCTTT
- TTTTTTGTTT

The running time for this code was 1:47min

3.4.5 For file yst08r.txt

When k=12, the output motifs were :

- TATTTTACTTCT
- TATTTTATTTTT
- TATTTTTTTTTT
- TATTTTTTTTTT
- TATCTTTTTTTC
- TATTTTTTATTT
- AATTTTTTTTTT
- TATTTTTTTTTT
- TATCTTTTATGT
- TAATTTTTTTTT
- TATTTTTCTTTT

The running time for this code was 1:54min

3.4.6 For file yst08r.txt

When k=15, the output motifs were :

- TTTACTTCTTTCTTG
- TTTATTTTTCTTTTC
- TTTATTTACCTATCT
- TTTATTATTCTCTCT
- TTTATTTCTTTCTTT
- TATACTTTTCTATTT
- TTGATTTTTTTCTCT
- TCTATTTTTCTCTCT
- TTTATTTCTCTCTCT
- TTTCTTTTTCTCTTT
- AATATTTTTCTTTTT

The running time for this code was 2:26min

Chapter 4

Conclusion

We can conclude by saying that both the tools perform exceptionally well when it comes to estimating motifs in a provided DNA. The algorithms that we have implemented i.e. Randomized Motif Search and Gibbs Sampling also produced convincing outputs. However, the RMS algoritm needs to be run multiple times for ensuring close to accurate answers.

Chapter 5

Reference

5.1 Method 1 : Randomized Motif Search

Why does randomized motif search work?.

5.2 Method 2 : Gibbs Sampler

Gibbs Sampler for Sequence Motif Detection Example.

5.3 Softtware

1. STREME
2. MEME-CHIP