CSE 463
Introduction to Bioinformatics
Assignment Report
on
Comparison Between Motif Finding Algorithms and Tools

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BANGLADESH UNIVERSITY OF

ENGINEERING AND TECHNOLOGY

Submitted To:
Dr. Muhammad Ali Nayeem
Assistant Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

SECTION: A

Assignment GROUP: 17

MEMBERS:

1905037 Syed Tehjeebuzzaman

1905039 Mustafa Siam Ur Rafique

190540 Asif Al Shahriar

1905041 Abdullah Al - Mohaimin

1905058 Sadif Ahmed

SUBMISSION DATE: 14/03/2024

Contents

1 Introduction

Motif finding, in the context of Bioinformatics, refers to the process of identifying short, recurring patterns (motifs) within biological sequences, most commonly DNA or protein sequences. These motifs are believed to have specific biological functions, such as:

- **Gene regulation:** Motifs can act as binding sites for transcription factors, proteins that control gene expression by attaching to specific DNA sequences.
- **Protein-protein interactions:** Motifs can be recognition sites where proteins bind to each other, enabling various cellular processes.
- **Protein function:** Motifs can be signatures for functional domains within proteins, hinting at their roles in the cell.

There are many algorithms and tools designed to find motif from given DNA sequences. In this assignment we have choose two such algorithms named Greedy Algorithm and Enumeration and Scoring Algorithm along with some modifications to each of the algorithms and tried to compare them with two SOTA tools in the motif finding field names MEME-SUITE and STREME and compared their scores.

2 Data

In this section We will give a brief about the datasets we have used for this comparison. We were provided with 3 unique datasets named dataset 1("hm03.txt"), dataset 2("yst04r.txt") and dataset 3 ("yst08r.txt"). Each of them contained 10,7 and 11 DNA sequences. We have run our modified algorithms and the chosen tools on these datasets and stored the results.

2.1 Bio-Marker

In our given assignment, we were not provided with any ground truth or reference motifs. So, We have decided on a universal scoring algorithm which relies on the Log likelihood Ratio concept of motif scoring. In our comparison section we see the algorithm and its usage in action. We are currently adding an overview of the algorithm in this section.

```
def calculate_background(sequences):
    background = Counter()
    for sequence in sequences:
        background.update(sequence)
    total_count = sum(background.values())
```

```
    return {nucleotide: count / total_count for
       nucleotide, count in background.items()}

def score_kmer_background(kmer, background):
    score = 0
    for nucleotide in kmer:
        score += np.log2( (1 / background[nucleotide
           ]) )  # Log-likelihood score
    return score
```

The above mentioned scoring algorithms might not give better scores than MEME or STREME found motifs as the algorithms used in the backend of SOTA tools are much more complex than our compared algorithms.

3 Methods

3.1 Greedy Motif Search

This algorithm iteratively builds a consensus motif by identifying the most frequent k-mer (substring of length k) in a set of sequences and adding it to the motif. It then removes sequences that don't contain the identified k-mer and repeats the process until a stopping criterion is met (e.g., reaching a specific motif length or no improvement in motif score).

3.1.1 Weighted scoring:

Instead of simply considering the most frequent k-mer, we have assigned weights to positions based on nucleotide frequency or conservation. This can help identify motifs with specific nucleotide preferences at certain positions.

```
def calculate_background(sequences):
    background = Counter()
    for sequence in sequences:
        background.update(sequence)
    total_count = sum(background.values())
    return {nucleotide: count / total_count for
       nucleotide, count in background.items()}

def score_kmer_background(kmer, background):
    score = 0
    for nucleotide in kmer:
        score += np.log2( (1 / background[nucleotide
           ]) )  # Log-likelihood score
    return score
```

```
def find_greedy_motif_background(sequences, k,
   max_iterations=100):

    background = calculate_background(sequences)

    best_motif = None
    best_score = float('-inf')

    for _ in range(max_iterations):
        print(_)
        motif_scores = {}
        for sequence in sequences:
            for i in range(len(sequence) - k + 1):
                kmer = sequence[i:i+k]
                motif_scores[kmer] =
                   score_kmer_background(kmer,
                   background)

        if not motif_scores:  # Check if motif_scores
            is empty
            break  # Terminate if no k-mers meet the
               conditions
        # Find the highest scoring k-mer
        best_kmer, score = max(motif_scores.items(),
           key=lambda item: item[1])
        if score > best_score:
            best_motif = best_kmer
            best_score = score

        # Remove sequences containing the motif
        sequences = [seq for seq in sequences if
           best_kmer not in seq]

    return best_motif, best_score
```

3.1.2 Randomization:

We have introduced randomization to explore different motif possibilities. Starting with a random sequence and iteratively refining it by replacing positions based on the most frequent k-mers in the remaining sequences.

```
def score_kmer(kmer, background):

    score = 0
```

```
    for nucleotide in kmer:
        score += np.log2( (1 / background[nucleotide
           ]) )  # Log-likelihood score
    return score

def find_greedy_motif_randomized(sequences, k,
   max_iterations=100):

    # Randomly initialize a motif
    motif = ''.join(random.choice('ACGT') for _ in
       range(k))

    best_motif = motif
    best_score = float('-inf')

    for _ in range(max_iterations):
        print(_)
        motif_scores = {}
        for sequence in sequences:
            for i in range(len(sequence) - k + 1):
                kmer = sequence[i:i+k]
                motif_scores[kmer] = score_kmer(kmer,
                    calculate_background(sequences))

        # Find the highest scoring k-mer (excluding
           the current motif)
        best_kmer, score = max(motif_scores.items(),
           key=lambda item: item[1])
        if score > best_score and best_kmer != motif:
            best_motif = best_kmer
            best_score = score

        # Randomly replace a position in the motif
           with the highest scoring k-mer
        replace_index = random.randint(0, k-1)
        motif = motif[:replace_index] + best_kmer[
           replace_index] + motif[replace_index+1:]

    return best_motif, best_score

def calculate_background(sequences):

    background = Counter()
    for sequence in sequences:
```

```
        background.update(sequence)
    total_count = sum(background.values())
    return {nucleotide: count / total_count for
       nucleotide, count in background.items()}
```

3.1.3 Multiple motif discovery:

We have extended the algorithm to find a set of k motifs instead of just one. This can be useful for identifying co-occurring motifs or multiple binding sites for different factors.

```
def find_greedy_motifs(sequences, k, num_motifs):

    motifs = []
    remaining_sequences = sequences.copy()

    for _ in range(num_motifs):
        print(_)
        # Find the motif in the remaining sequences
        motif, score =
           find_greedy_motif_single_randomized(
           remaining_sequences, k)
        motifs.append((motif, score))

        # Remove sequences containing the motif from
           remaining sequences
        remaining_sequences = [seq for seq in
           remaining_sequences if motif not in seq]

        # If no sequences remain for further motif
           discovery, terminate
        if not remaining_sequences:
            break
    return motifs
```

3.2 Enumeration and Scoring

This method explores all possible k-mers of a specific length and scores them based on their frequency and conservation in the sequences. The highest-scoring k-mer is considered the potential motif.

```
def generate_all_kmers(sequence, k):

    return set([sequence[i:i+k] for i in range(len(
       sequence) - k + 1)])
```

```
def calculate_kmer_score(kmer, sequences):

    count = sum(kmer in seq for seq in sequences)
    return count

def find_motif_kmer_scanning(sequences, k):

    kmer_scores = {}
    for sequence in sequences:
        for kmer in generate_all_kmers(sequence, k):
            kmer_scores[kmer] = calculate_kmer_score(
               kmer, sequences)

    best_kmer, best_score = max(kmer_scores.items(),
       key=lambda item: item[1])
    return best_kmer, best_score
```

3.2.1 Position-specific scoring matrices (PSSMs):

Instead of a simple frequency score, we utilized PSSMs that capture the probability of each nucleotide at each position in the motif. This allows for more nuanced scoring based on sequence conservation.

```
def calculate_pwm(sequences, k):

    pwm = np.zeros((4, k))  # 4 for A, C, G, T
    for sequence in sequences:
        for i in range(len(sequence) - k + 1):
            kmer = sequence[i:i+k]
            for j, nucleotide in enumerate(kmer):
                pwm[get_nucleotide_index(nucleotide),
                    j] += 1
    # Normalize to probabilities
    pwm = pwm / (len(sequences) * k)
    return pwm

def get_nucleotide_index(nucleotide):
    return {"A": 0, "C": 1, "G": 2, "T": 3}[
       nucleotide]

def score_kmer_pwm(kmer, pwm):
```

```
    score = 1.0  # Initialize with 1 to avoid
       underflow
    for i, nucleotide in enumerate(kmer):
        score *= pwm[get_nucleotide_index(nucleotide)
           , i]
    return np.log2(score)  # Log-likelihood score

def find_greedy_motif_pwm(sequences, k,
   max_iterations=100):

    best_motif = None
    best_score = float('-inf')

    for _ in range(max_iterations):
        pwm = calculate_pwm(sequences, k)

    # Initialize motif with the highest scoring k-mer
        based on the PWM
    motif_scores = {}
    for sequence in sequences:
        for i in range(len(sequence) - k + 1):
            kmer = sequence[i:i+k]
            motif_scores[kmer] = score_kmer_pwm(kmer,
                pwm)
    initial_motif, _ = max(motif_scores.items(), key=
       lambda item: item[1])

    # Iteratively refine the motif
    motif = initial_motif
    while True:
        motif_scores = {}
        for sequence in sequences:
            for i in range(len(sequence) - k + 1):
                kmer = sequence[i:i+k]
                motif_scores[kmer] = score_kmer_pwm(
                   kmer, pwm)

        best_kmer, score = max(motif_scores.items(),
           key=lambda item: item[1])
        if score > best_score:
            best_motif = best_kmer
            best_score = score
            motif = best_kmer  # Update for the next
               iteration
```

```
        else:
            break  # No improvement, stop refining

    return best_motif, best_score
```

3.2.2 Background model:

In this modification, we considered incorporating a background model that reflects the expected nucleotide frequencies in the absence of a specific motif. This helps distinguish true motifs from random occurrences.

```
def calculate_background(sequences):
    background = Counter()
    for sequence in sequences:
        background.update(sequence)
    total_count = sum(background.values())
    return {nucleotide: count / total_count for
       nucleotide, count in background.items()}

def score_kmer_background(kmer, background):
    score = 0
    for nucleotide in kmer:
        score += np.log2( (1 / background[nucleotide
           ]) )  # Log-likelihood score
    return score

def find_greedy_motif_background(sequences, k,
   max_iterations=100):

    background = calculate_background(sequences)

    best_motif = None
    best_score = float('-inf')

    for _ in range(max_iterations):
        print(_)
        motif_scores = {}
        for sequence in sequences:
            for i in range(len(sequence) - k + 1):
                kmer = sequence[i:i+k]
                motif_scores[kmer] =
                   score_kmer_background(kmer,
                   background)
```

```
        if not motif_scores:  # Check if motif_scores
            is empty
            break  # Terminate if no k-mers meet the
               conditions
        # Find the highest scoring k-mer
        best_kmer, score = max(motif_scores.items(),
           key=lambda item: item[1])
        if score > best_score:
            best_motif = best_kmer
            best_score = score

        # Remove sequences containing the motif
        sequences = [seq for seq in sequences if
           best_kmer not in seq]

    return best_motif, best_score
```

3.2.3 Thresholding and refinement:

We implemented a threshold for the minimum score to filter out low-scoring k-mers and focused on potentially significant motifs. We can then refine the top-scoring k-mers by iteratively replacing positions based on surrounding sequences.

```
def generate_all_kmers(sequence, k):
    return set([sequence[i:i+k] for i in range(len(
       sequence) - k + 1)])

def calculate_kmer_score(kmer, sequences):

    count = sum(kmer in seq for seq in sequences)
    return count

def find_motif_kmer_scanning_refined(sequences, k,
   min_score=0):

    kmer_scores = {}
    for sequence in sequences:
        for kmer in generate_all_kmers(sequence, k):
            score = calculate_kmer_score(kmer,
               sequences)
            if score >= min_score:  # Apply threshold
                kmer_scores[kmer] = score
```

```
if not kmer_scores:
    return "", 0  # No k-mers met the threshold

# Refine the top-scoring k-mer
best_kmer, best_score = max(kmer_scores.items(),
   key=lambda item: item[1])
for _ in range(10):  # Limit refinement
   iterations (optional)
    new_kmer = best_kmer
    for i in range(k):
        neighbor_kmers = {kmer[:i] + c + kmer[i
           +1:] for c in 'ACGT' if c != best_kmer
           [i]}
        neighbor_scores = {kmer:
           calculate_kmer_score(kmer, sequences)
           for kmer in neighbor_kmers}
        best_neighbor, neighbor_score = max(
           neighbor_scores.items(), key=lambda
           item: item[1])
        if neighbor_score > best_score:
            best_kmer = best_neighbor
            best_score = neighbor_score
            break  # Stop replacing if no
               improvement

return best_kmer, best_score
```

4 Software Tool Usage(MEME & STREME)

Here we are using two State Of The Art motif searching tools named MEME[2] and STREME[1]. Both of the tools provide a online GUI to operate the tools remotely by using their servers. The links are provided bellow:

- MEME
- STREME

But the queue time of their servers are quite substantial. So, we cloned their repository and ran the tools in our local machines which gave us much faster results and helped us achieve much more diversity in our experiments through exploring different parameters. The following two segments contains the shell codes used to run the tools and generate the result CSVs in the Code/results folder of the attached github repository.

4.1 Using MEME

Listing 1: MEME Shell Script

```
#!/bin/bash

# delete previous outputs
rm *fasta.txt
rm meme_output_smaller.csv
rm meme_outputs_smaller.zip

# create output csv file
echo "Input␣file,K,Motif,E␣Value,Time,Objective␣
   Function" >>meme_output_smaller.csv

# loop over
for target in $(ls *fasta); do
    c_=0
    # meme
    for i in {8..15}; do
        for objf_n in cd de; do
            ~/bin/meme ${target} -nostatus -objfun ${
               objf_n} -dna -w ${i} -alph
               dna_alphabet.txt -text >meme_out_s_k_$
               {i}_${c_}__${target}.txt
            ((c_++))

        done

        # many command line combinations are not
           eligible for output, delete these 0 size
           files
        find -size 0 -delete

        # this should work on 5 * 2 * 3 * 3 = 90
           files
        for file_target in $(ls meme_out_s*); do
            # source file name from output filename
            file_name=$(echo "$file_target" | grep -o
                -E "[a-z0-9]+\.fasta")

            # motif name(this is a combined name ,i.e
               . all combinations of neucleotides
            # have a name, N and X are wildcard) from
```

```
            output file
        motif=$(grep -o -G -m 1 "MOTIF␣[
            ATGCRYKMSWBDHVNX\?]*" $file_target |
            sed -r "s/MOTIF[␣]*//g")

        # width of motif from motif metadata
        w=$(grep -o -E "width␣=[␣]*[0-9]+"
            $file_target | sed -r "s/width[␣]*=[␣
            ]*//g")

        # time is on another line
        time_spent=$(grep "Time" $file_target |
            sed -r -e "s/Time[␣]*//g" -e "s/[␣]*
            secs.//g" | tail -n 1)

        # there is usually no score in meme
            output, rather an E-Value
        evalue=$(grep -o -E "E\-value␣=␣[\-]*[0-9
            e\.]*[\+\-]?[0-9]*" $file_target | sed
             -r "s/E\-value[␣]*=[␣]*//g")

        # following 1 value are collected from
            the command line arguments line
        obj_fun=$(grep -o -E "\-objfun[␣]*[a-z]+"
             $file_target | sed -r "s/\-objfun[␣
            ]*//g")

        echo "$file_name,$w,$motif,$evalue,
            $time_spent,$obj_fun" >>
            meme_output_smaller.csv
    done

    # zip all outputs and add to archive
    find . -name "meme_out_s*" -print | zip -q
        meme_outputs_smaller -@

    # delete meme outputs, clear folder a little
        bit for next step
    find -name "meme_out_s*" -delete

    #clear screen
    clear
  done
done
```

```
echo "done␣job"
```

4.2 Using STREME

Listing 2: STREME Shell Script

```
#!/bin/bash

# delete previous outputs
rm *fasta.txt
rm streme_output.csv
rm streme_outputs.zip

for target in $(ls *fasta); do
        for i in {8..15}; do
                for objf_n in cd de; do
                        ~/bin/streme --p ${target} --
                           verbosity 1 --objfun ${
                           objf_n} --w $i --nmotifs 1
                            --alph dna_alphabet.txt
                           --text >streme_out_k_${i}
                           _${objf_n}__${target}.txt
                done
        done
done

# many command line combinations are not eligible for
    output, delete these 0 size files
find -size 0 -delete

echo "Input␣file,K,Motif,Score,Time,Objective␣
   Function" >>streme_output.csv

for target in $(ls streme_out_*); do
        # source file name from output filename
        filename=$(echo "$target" | grep -o -E "[a-z0
           -9]+\.fasta")
        # motif name(this is a combined name ,i.e.
           all combinations of neucleotides
        # have a name, N and X are wildcard) from
           output file
        motif=$(grep -o -G "1\-[ATGCRYKMSWBDHVNX]*"
           $target | sed -n -r "s/1\-//p")
        # width of motif from the same line
```

```
        w=$(grep -o -E "w=␣[0-9]+" $target | sed -r -
           n "s/w=[␣]*//p")
        # time is in another line
        time_spent=$(grep "FINALTIME" $target | sed -
           n -r -e "s/FINALTIME[␣]*://p" -e "s/[␣]*
           seconds//p" | tail -n 1)
        # score is in same line of motif
        score=$(grep -o -E "S=␣[\-]*[0-9e
           \.]*[\+\-]?[0-9]*" $target | sed -n -r "s/
           S=[␣]*//p")
        # object function from output file name
        obj_fun=$(echo "$target" | grep -o -E "_[cde
           ]+_" | sed "s/_//g")

        echo "$filename,$w,$motif,$score,$time_spent,
           $obj_fun" >>streme_output.csv
done

# zip all outputs
find . -name "streme_out_*" -print | zip -q
   streme_outputs -@

# delete streme outputs, clear folder
find -name "streme_out_*" -delete

echo "done␣job"
```

5 Results

In our experiment we have used the previously described dataset 1,2 and 3 for the motif finding algorithms and their results are shown for K-mer size 8-15 in the tables 1-21.
In terms of tool usage we have converted the three datasets into their respective fasta formatted files ti meet the tool input requirements. The results obtained from the MEME and STREME tools while changing the only common parameter **"Objective Fuction"** and motif size K from 8-15 is shown in table 22-23.
In our final segment of this section we have accumulated a comparison of scores using the previously declared log likelihood scoring function to show the accuracy and effectiveness of the given algorithms in comparison with the tool. The results are shown in table 24,25,26 one for each dataset for a fixed k-mer size.

Input file	K	Motif	Score	Time
hm03.txt	8	AAAAAAAA	684	1.2870008945465088
hm03.txt	9	AAAAAAAAA	540	0.7739324569702148
hm03.txt	10	AGCTTAGTGC	465.0	0.797450065612793
hm03.txt	11	CAGCTTAGTGC	557.5	0.8693239688873291
hm03.txt	12	AATCAGCACTCT	624.0	0.33712053298950195
hm03.txt	13	CAATCAGCACTCT	718.0	0.34857678413391113
hm03.txt	14	AATCAGCACTCTGT	818.0	0.36506009101867676
hm03.txt	15	CAATCAGCACTCTGT	926.0	0.3871946334838867

Table 1: Weighted Greedy algorithm at dataset 1

Input file	K	Motif	Score	Time
yst04r.txt	8	AAAAAAAA	1188	0.7147562503814697
yst04r.txt	9	AAAAAAAAA	1215	0.497882604598999
yst04r.txt	10	AAAAAAAAAA	1210	0.35774993896484375
yst04r.txt	11	AAAAAAAAAAA	1188	0.36803722381591797
yst04r.txt	12	AAAAAAAAAAAA	1092	0.20643401145935059
yst04r.txt	13	AAAAAAAAAAAAA	1001	0.21043848991394043
yst04r.txt	14	AAAAAAAAAAAAAA	945	0.2260279655456543
yst04r.txt	15	AAAAAAAAAAAAAAA	960	0.234419584274292

Table 2: Weighted Greedy algorithm at dataset 2

Input file	K	Motif	Score	Time
yst08r.txt	8	TTTTTTTT	2268	1.0402636528015137
yst08r.txt	9	TTTTTTTTT	1620	0.7849507331848145
yst08r.txt	10	AAAAAAAAAA	1375	0.6957921981811523
yst08r.txt	11	AAAAAAAAAAA	1254	0.547152042388916
yst08r.txt	12	AAAAAAAAAAAA	1092	0.20844578742980957
yst08r.txt	13	AAAAAAAAAAAAA	1001	0.22543621063232422
yst08r.txt	14	AAAAAAAAAAAAAA	945	0.23637151718139648
yst08r.txt	15	TATATATATATATAT	992	0.24734210968017578

Table 3: Weighted Greedy algorithm at dataset 3

Input file	K	Motif	Score	Time
hm03.txt	8	CCGCCGCC	16.975755177729038	807.9076118469238
hm03.txt	9	CCGCCGCCC	19.10780601351023	790.5003533363342
hm03.txt	10	CCCGCCGCCC	21.239856849291424	820.0642943382263
hm03.txt	11	CCCGCCGCCCG	23.331581930812366	793.0314118862152
hm03.txt	12	CCCGCCGCCCGC	25.46363276659356	827.4436371326447
hm03.txt	13	CCCGCCGCCCGCC	27.595683602374752	821.5370583534241
hm03.txt	14	CCCGCCGCCCGCCC	29.727734438155945	850.8508081436157
hm03.txt	15	CCCGCCGCCCGCCCG	31.819459519676887	844.0370509624481

Table 4: Random Greedy algorithm at dataset 1

Input file	K	Motif	Score	Time
yst04r.txt	8	GCCGCGGG	19.409477181817884	164.88099932670593
yst04r.txt	9	GCCGCGGGC	21.707896442011094	163.4181830883026
yst04r.txt	10	GGCGGAGGGG	24.01949904773366	170.32040143013
yst04r.txt	11	GGCGGAGGGGA	25.717827793292898	163.73021960258484
yst04r.txt	12	GGCGGAGGGGAG	28.22067167354055	167.28891372680664
yst04r.txt	13	GGTGGCGGAGGGG	30.704908450560694	162.26219296455383
yst04r.txt	14	AGGTGGCGGAGGGG	32.40323719611993	170.03656005859375
yst04r.txt	15	GGTGGCGGAGGGGAG	34.906081076367585	168.7008922100067

Table 5: Random Greedy algorithm at dataset 2

Input file	K	Motif	Score	Time
yst08r.txt	8	GCCGCGGG	19.054988585741064	419.51497411727905
yst08r.txt	9	GCCGCGGGC	21.302784951051212	391.6183364391327
yst08r.txt	10	GGCGCCGCGC	23.55058131636136	419.30572390556335
yst08r.txt	11	GGCGCCGCGCC	25.798377681671507	398.8351631164551
yst08r.txt	12	GGCGCCGCGCCG	28.260697579633632	419.36436676979065
yst08r.txt	13	GGCGCCGCGCCGC	30.50849394494378	397.5376853942871
yst08r.txt	14	GGCGCCGCGCCGCC	32.75629031025393	419.48921155929565
yst08r.txt	15	GGCGCCGCGCCGCCG	35.218610208216056	390.9270529747009

Table 6: Random Greedy algorithm at dataset 3

Input file	K	Motif	Score	Time
hm03.txt	8	GCCGCGC	25.361817182970398	570.1238237857818
hm03.txt	8	GCTCAGC	25.063290876773582	570.1238237857818
hm03.txt	8	GGAGGCT	24.96819385685265	570.1238237857818
hm03.txt	8	CTGCTCA	24.855224284797167	570.1238237857818
hm03.txt	8	AGCACTG	24.8301577763892	570.1238237857818
hm03.txt	9	GGCCGCGC	27.454503740728782	563.923824596405
hm03.txt	9	CGGAGGCT	27.10475559560504	563.923824596405
hm03.txt	9	GCTGCTCA	27.005747218092907	563.923824596405
hm03.txt	9	CAGCACTG	27.01407357464191	563.923824596405
hm03.txt	9	AGCTCAGC	26.95908272039182	563.923824596405
hm03.txt	10	CGGCCGCGC	29.587516052747418	537.5639543056488
hm03.txt	10	CCGGAGGCT	29.288199926615626	537.5639543056488
hm03.txt	10	CGCTGCTCA	29.203206097768355	537.5639543056488
hm03.txt	10	CAGCTCAGC	29.22911588905331	537.5639543056488
hm03.txt	10	GGCGTGGAA	29.155467917430673	537.5639543056488
hm03.txt	11	CCGGCCGCGC	31.72052836476606	520.3948437213897
hm03.txt	11	GCCGGAGGCT	31.400654451536646	520.3948437213897
hm03.txt	11	GCGCTGCTCA	31.34128643036487	520.3948437213897
hm03.txt	11	GCAGCTCAGC	31.381775158757755	520.3948437213897
hm03.txt	11	GGGCGTGGAA	31.33576252252285	520.3948437213897
hm03.txt	12	GCCGGCCGCGC	33.813214922524445	535.7719171524047
hm03.txt	12	CGCCGGAGGCT	33.58409878254723	535.7719171524047
hm03.txt	12	GGCGCTGCTCA	33.47936676296139	535.7719171524047
hm03.txt	12	CGGGCGTGGAA	33.409668339048295	535.7719171524047
hm03.txt	12	CCACAGCACTG	33.74037692871948	535.7719171524047
hm03.txt	13	CGCCGGCCGCGC	35.94622723454309	517.1254944324494
hm03.txt	13	GCGCCGGAGGCT	35.69655330746825	517.1254944324494
hm03.txt	13	GCGGGCGTGGAA	35.43970777900651	517.1254944324494
hm03.txt	13	CCCACAGCACTG	35.82098429169355	517.1254944324494
hm03.txt	13	CTGCAGCTCAGC	35.803565146949595	517.1254944324494
hm03.txt	14	GCGCCGGCCGCGC	38.03891379230147	531.313570022583
hm03.txt	14	GGCGCCGGAGGCT	37.80900783238927	531.313570022583
hm03.txt	14	CGCGGGCGTGGAA	37.637166658681956	531.313570022583
hm03.txt	14	CCCCACAGCACTG	38.11194099951307	531.313570022583
hm03.txt	14	CAGGCGCTGCTCA	38.016908423872245	531.313570022583
hm03.txt	15	GGCGCCGGCCGCGC	40.13160035005986	511.50658240318296
hm03.txt	15	GGGCGCCGGAGGCT	39.92146235731029	511.50658240318296
hm03.txt	15	GCCCCACAGCACTG	39.697193249486276	511.50658240318296
hm03.txt	15	GCAGGCGCTGCTCA	39.52605919610705	511.50658240318296
hm03.txt	15	TCGCGGGCGTGGAA	39.48187611460287	511.50658240318296

Table 7: Multiple Greedy algorithm at dataset 1

Input file	K	Motif	Score	Time
yst04r.txt	8	CTAGAGA	24.05613497775129	103.7783534526825
yst04r.txt	8	CATCACA	23.635529830006718	103.7783534526825
yst04r.txt	8	TAAAACA	22.58090268036199	103.7783534526825
yst04r.txt	8	ACTAAAA	22.55075006067379	103.7783534526825
yst04r.txt	8	ATATACA	22.51084342887703	103.7783534526825
yst04r.txt	9	GCTAGAGA	26.560420832172852	95.84646973609924
yst04r.txt	9	TCATCACA	25.302022216224472	95.84646973609924
yst04r.txt	9	CATATACA	24.787186089613677	95.84646973609924
yst04r.txt	9	AACTAAAA	24.275229303237687	95.84646973609924
yst04r.txt	9	ATAAAACA	24.220485943421895	95.84646973609924
yst04r.txt	10	AGCTAGAGA	28.260191551905994	104.17068257331849
yst04r.txt	10	ATCATCACA	27.030047766037463	104.17068257331849
yst04r.txt	10	TCATATACA	26.476427601149904	104.17068257331849
yst04r.txt	10	AAACTAAAA	25.999281578600726	104.17068257331849
yst04r.txt	10	TATAAAACA	25.92718465203467	104.17068257331849
yst04r.txt	11	AAGCTAGAGA	29.95996227163914	98.58565678596497
yst04r.txt	11	CATCATCACA	29.30262594454213	98.58565678596497
yst04r.txt	11	CTCATATACA	28.728035000911085	98.58565678596497
yst04r.txt	11	CTAAATCAAA	28.23891869546112	98.58565678596497
yst04r.txt	11	TAAACTAAAA	27.770694398562995	98.58565678596497
yst04r.txt	12	AAAGCTAGAGA	31.659732991372284	99.36925983428955
yst04r.txt	12	ACATCATCACA	31.03065149435512	99.36925983428955
yst04r.txt	12	TCTCATATACA	30.41727651244731	99.36925983428955
yst04r.txt	12	ACTAAATCAAA	29.962970970824163	99.36925983428955
yst04r.txt	12	ATAAACTAAAA	29.513918983200885	99.36925983428955
yst04r.txt	13	TAAAGCTAGAGA	33.340896607877916	101.70585007667542
yst04r.txt	13	AACATCATCACA	32.75867704416811	101.70585007667542
yst04r.txt	13	CACTAAATCAAA	32.24248999551201	101.70585007667542
yst04r.txt	13	ATCTCATATACA	32.20341347880485	101.70585007667542
yst04r.txt	13	CATAAACTAAAA	31.883713389353506	101.70585007667542
yst04r.txt	14	ATAAAGCTAGAGA	35.04066732761106	105.86462502479553
yst04r.txt	14	AAACATCATCACA	34.4867025939811	105.86462502479553
yst04r.txt	14	ACACTAAATCAAA	33.97705549755773	105.86462502479553
yst04r.txt	14	TATCTCATATACA	33.833017835648334	105.86462502479553
yst04r.txt	14	CAAATATAAAACA	33.626937973991396	105.86462502479553
yst04r.txt	15	GATAAAGCTAGAGA	37.544953182032614	93.65210862159729
yst04r.txt	15	AAAACATCATCACA	36.21472814379409	93.65210862159729
yst04r.txt	15	CTATCTCATATACA	36.092690925790436	93.65210862159729
yst04r.txt	15	AACACTAAATCAAA	35.681334254151956	93.65210862159729
yst04r.txt	15	ACAAATATAAAACA	35.37016255862929	93.65210862159729

Table 8: Multiple Greedy algorithm at dataset 2

Input file	K	Motif	Score	Time
yst08r.txt	8	CATCACA	23.65921537297101	282.3894196033478
yst08r.txt	8	ATAAACG	23.38331221262603	282.3894196033478
yst08r.txt	8	ACTAAAC	23.20094773400905	282.3894196033478
yst08r.txt	8	AATTACC	23.121885797979644	282.3894196033478
yst08r.txt	8	AATCAAA	22.684561709780482	282.3894196033478
yst08r.txt	9	TCATCACA	25.346092640671042	248.91851201057435
yst08r.txt	9	CAATTACC	25.352645721464086	248.91851201057435
yst08r.txt	9	AATAAACG	25.07678394144385	248.91851201057435
yst08r.txt	9	CAATCAAA	24.937511219372855	248.91851201057435
yst08r.txt	9	AACTAAAC	24.96405130996117	248.91851201057435
yst08r.txt	10	CCAATTACC	27.595330980155094	258.1572421550751
yst08r.txt	10	GAATAAACG	27.547013312418738	258.1572421550751
yst08r.txt	10	ATCATCACA	27.106790884504278	258.1572421550751
yst08r.txt	10	TCAATCAAA	26.610116729317223	258.1572421550751
yst08r.txt	10	TAACTAAAC	26.647858436214015	258.1572421550751
yst08r.txt	11	CATCATCACA	29.3477965894827	247.47510623931885
yst08r.txt	11	AGAATAAACG	29.34585140029691	247.47510623931885
yst08r.txt	11	TCCAATTACC	29.25092334001924	247.47510623931885
yst08r.txt	11	CAATAACAAA	28.4445401408335	247.47510623931885
yst08r.txt	11	CTAAATCAAA	28.192825293551	247.47510623931885
yst08r.txt	12	GTCCAATTACC	31.745970119991156	258.2210346698761
yst08r.txt	12	ACATCATCACA	31.126746601965138	258.2210346698761
yst08r.txt	12	AAGAATAAACG	31.010201773755387	258.2210346698761
yst08r.txt	12	GCAATAACAAA	30.91291719313777	258.2210346698761
yst08r.txt	12	ACTAAATCAAA	29.91960092043701	258.2210346698761
yst08r.txt	13	TGTCCAATTACC	33.43284738769118	247.32640891075135
yst08r.txt	13	AACATCATCACA	32.86035318404143	247.32640891075135
yst08r.txt	13	TAAGAATAAACG	32.696657262460775	247.32640891075135
yst08r.txt	13	AGCAATAACAAA	32.66643165386809	247.32640891075135
yst08r.txt	13	CACTAAATCAAA	32.17946804719212	247.32640891075135
yst08r.txt	14	GTAAGAATAAACG	35.24876401542146	258.7231081008911
yst08r.txt	14	TTGTCCAATTACC	35.07082354652846	258.7231081008911
yst08r.txt	14	AAACATCATCACA	34.6253542706343	258.7231081008911
yst08r.txt	14	TAGCAATAACAAA	34.33903716381246	258.7231081008911
yst08r.txt	14	ACACTAAATCAAA	33.90624367407813	258.7231081008911
yst08r.txt	15	AGTAAGAATAAACG	37.00122962474906	248.09140582084655
yst08r.txt	15	TTTGTCCAATTACC	36.746648845880756	248.09140582084655
yst08r.txt	15	CTAGCAATAACAAA	36.58261518357776	248.09140582084655
yst08r.txt	15	AAAACATCATCACA	36.16406848711529	248.09140582084655
yst08r.txt	15	AACACTAAATCAAA	35.633019300964136	248.09140582084655

Table 9: Multiple Greedy algorithm at dataset 3

Input file	K	Motif	Score	Time
hm03.txt	8	AAAATAAA	6	0.5105018615722656
hm03.txt	9	AAACAAAAT	4	0.547950029373169
hm03.txt	10	GCCTGACACA	3	0.5320727825164795
hm03.txt	11	GGGTGTGGCTT	3	0.5528585910797119
hm03.txt	12	TTTAACTTAATA	3	0.5621905326843262
hm03.txt	13	CTTAGTGCCTGAC	3	0.5883514881134033
hm03.txt	14	CAGCTTAGTGCCTG	3	0.5662860870361328
hm03.txt	15	TTTAACTTAATATTT	3	0.5767054557800293

Table 10: Enumeration scoring at dataset 1

Input file	K	Motif	Score	Time
yst04r.txt	8	ATTTTTTT	6	0.1354539394378662
yst04r.txt	9	ATTTTTTTT	4	0.1326155662536621
yst04r.txt	10	TTTTTTTTCT	4	0.1383967399597168
yst04r.txt	11	CTTTTCTGGCA	3	0.13551712036132812
yst04r.txt	12	ATTAATTTTTTT	2	0.13797950744628906
yst04r.txt	13	TTTTTTTTTCTTT	2	0.1420302391052246
yst04r.txt	14	AAGGAAGAAAAAAA	2	0.14326214790344238
yst04r.txt	15	GAATTACTCGTGAGT	1	0.13610506057739258

Table 11: Enumeration scoring at dataset 2

Input file	K	Motif	Score	Time
yst08r.txt	8	ATTTTTTT	7	0.32741785049438477
yst08r.txt	9	AAGAAAAAA	6	0.35312724113464355
yst08r.txt	10	GAAAAAAAAA	5	0.3304011821746826
yst08r.txt	11	AAGAAAAAAAA	4	0.31923723220825195
yst08r.txt	12	AAGAAAAAAAAA	4	0.31566357612609863
yst08r.txt	13	AAGAAAAAAAAAG	2	0.3213965892791748
yst08r.txt	14	AAGGAAGAAAAAAA	2	0.32398176193237305
yst08r.txt	15	CACTAATTGAGCGAT	1	0.3276033401489258

Table 12: Enumeration scoring at dataset 3

Input file	K	Motif	Score	Time
hm03.txt	8	GCCGCGC	25.361817182970398	0.5615570545196533
hm03.txt	9	GGCCGCGC	27.454503740728782	0.6171197891235352
hm03.txt	10	CGGCCGCGC	29.587516052747418	0.6613306999206543
hm03.txt	11	CCGGCCGCGC	31.72052836476606	0.729912281036377
hm03.txt	12	GCCGGCCGCGC	33.813214922524445	0.8156647682189941
hm03.txt	13	CGCCGGCCGCGC	35.94622723454309	0.8333520889282227
hm03.txt	14	GCGCCGGCCGCGC	38.03891379230147	0.889373779296875
hm03.txt	15	GGCGCCGGCCGCGC	40.13160035005986	0.9234464168548584

Table 13: Basic background model at dataset 1

Input file	K	Motif	Score	Time
yst04r.txt	8	CTAGAGA	24.05613497775129	0.22733807563781738
yst04r.txt	9	GCTAGAGA	26.560420832172852	0.2341156005859375
yst04r.txt	10	AGCTAGAGA	28.260191551905994	0.26662254333496094
yst04r.txt	11	AAGCTAGAGA	29.95996227163914	0.28133606910705566
yst04r.txt	12	AAAGCTAGAGA	31.659732991372284	0.3133273124694824
yst04r.txt	13	TAAAGCTAGAGA	33.340896607877916	0.3189582824707031
yst04r.txt	14	ATAAAGCTAGAGA	35.04066732761106	0.3601109981536865
yst04r.txt	15	GATAAAGCTAGAGA	37.544953182032614	0.36545872688293457

Table 14: Basic background model at dataset 2

Input file	K	Motif	Score	Time
yst08r.txt	8	CATCACA	23.65921537297101	0.4511837959289551
yst08r.txt	9	TCATCACA	25.346092640671042	0.5395472049713135
yst08r.txt	10	CCAATTACC	27.595330980155094	0.616534948348999
yst08r.txt	11	CATCATCACA	29.3477965894827	0.5855958461761475
yst08r.txt	12	GTCCAATTACC	31.745970119991156	0.6283402442932129
yst08r.txt	13	TGTCCAATTACC	33.43284738769118	0.6602873802185059
yst08r.txt	14	GTAAGAATAAACG	35.24876401542146	0.7118191719055176
yst08r.txt	15	AGTAAGAATAAACG	37.00122962474906	0.7417125701904297

Table 15: Basic background model at dataset 3

Input file	K	Motif	Score	Time
hm03.txt	8	AAAAAAAA	45.409063332379816	20.56632137298584
hm03.txt	9	AAAAAAAAA	49.54969480241341	22.601773023605347
hm03.txt	10	AAAAAAAAAA	53.52722216747364	24.224986791610718
hm03.txt	11	AAAAAAAAAAA	57.35881987948691	25.74872875213623
hm03.txt	12	AAAAAAAAAAAA	61.05660351490079	25.81015110015869
hm03.txt	13	AAAAAAAAAAAAA	64.63295783839445	27.025185585021973
hm03.txt	14	AAAAAAAAAAAAAA	68.09680751755114	27.864421844482422
hm03.txt	15	AAAAAAAAAAAAAAA	71.45503640606758	37.67752647399902

Table 16: PSSM at dataset 1

Input file	K	Motif	Score	Time
yst04r.txt	8	TTTTTTTT	42.250138398639784	9.336209058761597
yst04r.txt	9	TTTTTTTTT	45.993632072604285	10.980934619903564
yst04r.txt	10	TTTTTTTTTT	49.57317295005166	12.482874393463135
yst04r.txt	11	ATTTTTTTTTT	52.97054837145139	12.920790672302246
yst04r.txt	12	TATTTTTTTTTT	56.26616939073346	14.211660623550415
yst04r.txt	13	TTATTTTTTTTTT	59.43739890813412	14.265238523483276
yst04r.txt	14	TTTATTTTATTTTT	62.46052756372471	14.86003565788269
yst04r.txt	15	TTTTATTTTATTTTT	65.41486282186229	15.223716259002686

Table 17: PSSM at dataset 2

Input file	K	Motif	Score	Time
yst08r.txt	8	TTTTTTTT	42.18484679903607	14.820258140563965
yst08r.txt	9	TTTTTTTTT	45.91850198007205	16.655802249908447
yst08r.txt	10	TTTTTTTTTT	49.48812354468567	17.45496964454651
yst08r.txt	11	TTTTTTTTTTT	52.90854831959463	18.813684463500977
yst08r.txt	12	TATTTTTTTTTT	56.11012920668569	21.09698724746704
yst08r.txt	13	TTATTTTTTTTTT	59.27187701990238	22.02561068534851
yst08r.txt	14	TTTTTATTTTATTT	62.24220095480782	23.638184309005737
yst08r.txt	15	TTTTATTTTTATTTT	65.18060996700466	27.250149250030518

Table 18: PSSM at dataset 3

Input file	K	Motif	Score	Time
hm03.txt	8	AAAATAAA	6	0.5645897388458252
hm03.txt	9	AAAAAAAAT	4	0.6085407733917236
hm03.txt	10	TACTGCCTCT	3	0.6239364147186279
hm03.txt	11	CAGCTTAGTGC	3	0.6592864990234375
hm03.txt	12	CAGCTTAGTGCC	3	0.6189296245574951
hm03.txt	13	TTAACTTAATATT	3	0.5707738399505615
hm03.txt	14	AGCTTAGTGCCTGA	3	0.5769529342651367
hm03.txt	15	CCCAGCTTAGTGCCT	3	0.6270692348480225

Table 19: Thresholding and refinement at dataset 1

Input file	K	Motif	Score	Time
yst04r.txt	8	ATTTTTTT	6	0.12940621376037598
yst04r.txt	9	TTTTTTTCT	4	0.14007043838500977
yst04r.txt	10	TTTTTTTTCT	4	0.13393759727478027
yst04r.txt	11	CTTTTCTGGCA	3	0.14027810096740723
yst04r.txt	12	ATTTTTATTTTA	2	0.13590097427368164
yst04r.txt	13	TTTTTTTTTCTTT	2	0.14274978637695312
yst04r.txt	14	AAGGAAGAAAAAAA	2	0.14658689498901367
yst04r.txt	15		0	0.1354691982269287

Table 20: Thresholding and refinement at dataset 2

Input file	K	Motif	Score	Time
yst08r.txt	8	ATTTTTTT	7	0.3055763244628906
yst08r.txt	9	AAAAAAAAA	6	0.3159162998199463
yst08r.txt	10	GAAAAAAAAA	5	0.34259867668151855
yst08r.txt	11	AAGAAAAAAAA	4	0.3502049446105957
yst08r.txt	12	AAGAAAAAAAAA	4	0.3225085735321045
yst08r.txt	13	AAGGAAGAAAAAA	2	0.3225407600402832
yst08r.txt	14	AAGGAAGAAAAAAA	2	0.319049596786499
yst08r.txt	15		0	0.32456183433532715

Table 21: Thresholding and refinement at dataset 3

Input file	K	Motif	Score	Time	Objective Function
hm03.fasta	8	ACTAGTTG	2.30E-06	" 0.30"	cd
yst04r.fasta	8	CACCTGGA	1.10E-05	" 0.16"	cd
yst08r.fasta	8	CACGATTT	1.00E-07	" 0.19"	cd
hm03.fasta	8	GACAGAAA	6.00E-05	" 0.28"	de
yst04r.fasta	8	AAGAAAAA	2.90E-04	" 0.22"	de
yst08r.fasta	8	AGAGAGAG	1.70E-05	" 0.25"	de
hm03.fasta	9	AGAAGCTGG	9.40E-08	" 0.25"	cd
yst04r.fasta	9	GCAATCACA	2.10E-06	" 0.21"	cd
yst08r.fasta	9	CTTCCACAA	9.30E-09	" 0.27"	cd
hm03.fasta	9	CTTTTCCAA	5.40E-06	" 0.34"	de
yst04r.fasta	9	AGGAAGAAA	2.90E-04	" 0.45"	de
yst08r.fasta	9	AAGAGAAAA	1.40E-06	" 0.42"	de
hm03.fasta	10	GTACACAGAG	1.00E-07	" 0.26"	cd
yst04r.fasta	10	GGTGGAAGAA	1.10E-05	" 0.20"	cd
yst08r.fasta	10	AGTGGAAGCT	1.00E-07	" 0.26"	cd
hm03.fasta	10	AAAGAAGAAA	5.40E-06	" 0.36"	de
yst04r.fasta	10	AAGGAAGAAA	2.90E-04	" 0.27"	de
yst08r.fasta	10	AAAAAAAAAA	1.40E-06	" 0.41"	de
hm03.fasta	11	GAATAAATGAT	4.50E-07	" 0.29"	cd
yst04r.fasta	11	CAAAAAAATGG	2.10E-06	" 0.21"	cd
yst08r.fasta	11	CTTCCAGTTTT	1.90E-08	" 0.35"	cd
hm03.fasta	11	AAAGAAGAAAG	5.40E-06	" 0.61"	de
yst04r.fasta	11	GAAAAGAAAAA	2.90E-04	" 0.32"	de
yst08r.fasta	11	AAAAAAAAATT	1.40E-06	" 0.34"	de
hm03.fasta	12	GACATTTGACAA	1.00E-07	" 0.31"	cd
yst04r.fasta	12	GAAGAAAAAAAT	1.10E-05	" 0.22"	cd
yst08r.fasta	12	CTCAAGAAAAAA	2.20E-08	" 0.33"	cd
hm03.fasta	12	AAACAAAATAAA	6.00E-05	" 0.48"	de
yst04r.fasta	12	ATTTTTCTTTTC	2.90E-04	" 0.36"	de
yst08r.fasta	12	AAAAAAAAAAAT	1.40E-06	" 0.50"	de
hm03.fasta	13	GGGTGCTCCTACC	4.50E-07	" 0.39"	cd
yst04r.fasta	13	AAAAGAAAAATAT	1.00E-05	" 0.28"	cd
yst08r.fasta	13	GAAGAAAAAAATA	9.10E-08	" 0.38"	cd
hm03.fasta	13	AAAACAAAGCGAA	6.00E-05	" 0.57"	de
yst04r.fasta	13	GAAAAGAAAAAAA	2.90E-04	" 0.51"	de
yst08r.fasta	13	AAAAAAAAAAATG	1.40E-06	" 0.50"	de
hm03.fasta	14	AAAAAAAAGTGAAA	4.80E-07	" 0.58"	cd
yst04r.fasta	14	GAAAAGAAAAAAAA	1.10E-05	" 0.25"	cd
yst08r.fasta	14	GGAAAAAAAAAAA	2.20E-08	" 0.32"	cd
hm03.fasta	14	AAAACAAAGCGAAG	6.00E-05	" 0.50"	de
yst04r.fasta	14	GAAAAGAAAAAAAA	2.90E-04	" 0.35"	de
yst08r.fasta	14	AAAAAAAAAAATAG	1.40E-06	" 0.50"	de
hm03.fasta	15	CCTGACACAGGGAGG	4.50E-07	" 0.42"	cd
yst04r.fasta	15	AAAAAGAAAAATA	1.00E-05	" 0.28"	cd
yst08r.fasta	15	GGAAAAAAAAAAA	1.90E-08	" 0.31"	cd
hm03.fasta	15	AAACAAAATAAATAC	1.50E-03	" 0.55"	de
yst04r.fasta	15	GAAAAGAAAAAAAAA	2.90E-04	" 0.41"	de
yst08r.fasta	15	AAAAAAAAAAATGAA	1.40E-06	" 0.65"	de

Table 22: STREME Tool Output

Input file	K	Motif	E Value	Time	Objective Function
hm03.fasta	8	TCACCAAA	4.90E-01	0.18	cd
hm03.fasta	8	AAATATTT	2.90E-01	0.34	de
yst04r.fasta	8	CTGTAGCC	2.60E-01	0.04	cd
yst04r.fasta	8	ATGTTGGT	4.00E-01	0.08	de
yst08r.fasta	8	ACC?ACGC	8.40E-01	0.05	cd
yst08r.fasta	8	GGCACGTG	4.00E-01	0.18	de
hm03.fasta	9	ATTTTCCTC	1.40E-01	0.09	cd
hm03.fasta	9	AAAAAAAAA	1.00E+00	0.34	de
yst04r.fasta	9	CTGC?TCAT	8.50E-02	0.03	cd
yst04r.fasta	9	CAAGAATAA	6.70E-01	0.08	de
yst08r.fasta	9	CTTCCGCCC	7.20E-01	0.05	cd
yst08r.fasta	9	AAAAAAAAA	4.00E-01	0.18	de
hm03.fasta	10	TCC?CCAAAG	1.30E+00	0.07	cd
hm03.fasta	10	AAAAAAAAAA	1.00E+00	0.35	de
yst04r.fasta	10	GTGGCGGAGG	3.20E-01	0.04	cd
yst04r.fasta	10	CCAAGAATA?	1.00E+00	0.09	de
yst08r.fasta	10	AAAAAAGGAG	2.40E-01	0.06	cd
yst08r.fasta	10	CAAGGAAGAA	8.30E-02	0.34	de
hm03.fasta	11	AATAAACATTT	5.00E-01	0.07	cd
hm03.fasta	11	GAAAAAAAAAT	8.90E-01	0.34	de
yst04r.fasta	11	TGA?TCATTG?	1.20E-01	0.03	cd
yst04r.fasta	11	AGAAGAAGTGA	6.70E-02	0.08	de
yst08r.fasta	11	ACACCCATGCA	4.60E-01	0.05	cd
yst08r.fasta	11	GGCA?GGG?GA	7.70E-01	0.19	de
hm03.fasta	12	CTTTTTACT?CA	7.60E-02	0.08	cd
hm03.fasta	12	AAAAAAAATAAA	1.10E-01	0.33	de
yst04r.fasta	12	CATTTA?ACTGT	2.40E-01	0.03	cd
yst04r.fasta	12	AAGAAGAAGTGA	6.70E-02	0.07	de
yst08r.fasta	12	T?AAAGATTACG	4.60E-01	0.05	cd
yst08r.fasta	12	CTTCCTCTTTT?	8.30E-02	0.16	de
hm03.fasta	13	TTTTCATCTTCAT	5.90E-01	0.07	cd
hm03.fasta	13	AAAAAAAAAAAAA	5.00E-01	0.34	de
yst04r.fasta	13	GAACTACACTGC?	2.30E-01	0.03	cd
yst04r.fasta	13	AAGAAGAAGTGAA	6.70E-02	0.08	de
yst08r.fasta	13	CGGGTTATCTTC?	2.80E-01	0.05	cd
yst08r.fasta	13	TTGTGAGGGAGAC	6.00E-01	0.16	de
hm03.fasta	14	TTTTCCTCTTCATC	3.50E-01	0.08	cd
hm03.fasta	14	GGTATTATTAAA?C	7.10E-01	0.41	de
yst04r.fasta	14	G?ATT?A?ACTGTT	7.20E-01	0.03	cd
yst04r.fasta	14	AAAGAAGAAGTGAA	6.70E-02	0.09	de
yst08r.fasta	14	AAGGGTT?T?TACT	1.30E+00	0.05	cd
yst08r.fasta	14	AATAAACACATACA	7.70E-01	0.19	de
hm03.fasta	15	TTTTTCCTCTTCATC	3.50E-01	0.08	cd
hm03.fasta	15	GAAAAAAAAATAAAT	8.90E-01	0.32	de
yst04r.fasta	15	CTACGTTACC?TGAT	5.20E-02	0.04	cd
yst04r.fasta	15	GAGAAAGAAGTTAAT	6.70E-02	0.09	de
yst08r.fasta	15	TT?GGCCGCTAATCG	5.20E-01	0.06	cd
yst08r.fasta	15	CAAAAAA?AAAAAGC	2.30E-01	0.22	de

Table 23: MEME Tool Output

Algorithm/Tool	K	Motif	Score
Weighted Greedy	14	AATCAGCACTCTGT	27.88389195
Randomised Greedy	14	CCCGCCGCCCGCCC	29.72773444
Multiple Greedy	14	GCGCCGGCCGCGC	27.47470634
Enumeration and Scoring	14	CAGCTTAGTGCCTG	28.32691923
PSSM	14	AAAAAAAAAAAAAA	26.18296018
Threshold Refining	14	AGCTTAGTGCCTGA	28.06507984
Background Model	14	GCGCCGGCCGCGC	27.47470634
MEME(cd)	14	TTTTCCTCTTCATC	27.91325098
MEME(de)	14	GGTATTATTAAAAC	27.15101049
STREME(cd)	14	AAAAAAAAGTGAAA	26.67862419
STREME(de)	14	AAAACAAAGCGAAG	27.37117989

Table 24: Score Comparison of Dataset 1

Algorithm/Tool	K	Motif	Score
Weighted Greedy	14	AAAAAAAAAAAAAA	23.77660244
Randomised Greedy	14	AGGTGGCGGAGGGG	32.4032372
Multiple Greedy	14	ATAAAGCTAGAGA	25.0546954
Enumeration and Scoring	14	AAGGAAGAAAAAAA	26.19014784
PSSM	14	TTTATTTTATTTTT	23.5533172
Threshold Refining	14	AAGGAAGAAAAAAA	26.19014784
Background Model	14	ATAAAGCTAGAGA	25.0546954
MEME(cd)	14	GAATTAAAACTGTT	25.89268771
MEME(de)	14	AAAGAAGAAGTGAA	26.97605587
STREME(cd)	14	GAAAAGAAAAAAAA	25.38563271
STREME(de)	14	GAAAAGAAAAAAAA	25.38563271

Table 25: Score Comparison of Dataset 2

Algorithm/Tool	K	Motif	Score
Weighted Greedy	14	AAAAAAAAAAAAAA	24.51433089
Randomised Greedy	14	GGCGCCGCGCCGCC	32.75629031
Multiple Greedy	14	GTAAGAATAAACG	25.26279209
Enumeration and Scoring	14	AAGGAAGAAAAAAA	26.64821968
PSSM	14	TTTTTATTTTATTT	23.72727079
Threshold Refining	14	AAGGAAGAAAAAAA	26.64821968
Background Model	14	GTAAGAATAAACG	25.26279209
MEME(cd)	14	AAGGGTTATATACT	26.8170507
MEME(de)	14	AATAAACACATACA	25.8734724
STREME(cd)	14	GGAAAAAAAAAAAA	25.93692342
STREME(de)	14	AAAAAAAAAAATAG	25.16003881

Table 26: Score Comparison of Dataset 3

6 Conclusion

Before drawing conclusion, we would first like to say that the scoring function used in this report is a very simple one and it is not preferred to be used as a mode of evaluating complex tools such as MEME and STREME. Moreover, the scoring method is also the backbone of our greedy algorithms which makes its use a bit biased towards the greedy algorithms.
If we were to look at the overall result we can see that the randomised greedy algorithm is performing on par and sometimes better than the tools. So, in terms of algorithm the randomised greedy algorithm is the better choice given our test datasets and in terms of tools we can see that the scores are pretty similar and if we were to choose one we would choose MEME over STREME for smaller motif sizes.

References

[1] Timothy L Bailey. "STREME: accurate and versatile sequence motif discovery". In: *Bioinformatics* 37.18 (2021), pp. 2834–2840.

[2] Timothy L Bailey, Charles Elkan, et al. "Fitting a mixture model by expectation maximization to discover motifs in bipolymers". In: (1994).