SEARCHING FOR MOTIFS IN SEQUENCES

Python and Unix for Bioinformaticians - 22110

s202274 & s202302

Denmarks Technical University

INTRODUCTION

Sequence motifs are recurring, short patterns in DNA that presumably have a biological function. These functions are quite variable and range from specific binding sites for transcription factors or nucleases, to RNA level processes as ribosome binding, mRNA processing or transcription termination (1).

Our project presents a program capable of finding a given sequence motif within a sequence with a requested accuracy. The accuracy variability is important for different reasons:

- Regulatory proteins can bind degenerated consensus sequences, meaning that a single protein can bind different motifs which are similar but not exactly equal. Sometimes this conservation of the sequence with respect to the consensus one, correlates with the activity of the regulated gene, in other words; a higher difference will mean a weaker binding, reducing the possibilities of an interaction event (1).

- Known regulatory motifs can vary between regulatory proteins or species, thus; close ones (in terms of evolution) will present more similar sequence motifs and a higher similarity will be demanded to find its presence in homologous systems.

Even though our program is focused on a DNA motif sequence search, it has been designed to be perfectly capable of being adapted and used in other fields.

CONTRIBUTION

Content	s202274	S202302
Report	50%	50%
Program code	50%	50%
Data files of relevance	50%	50%

THEORY

Computational methods are commonly used as they are capable to perform high-throughput screening, this screening has become so extensive that nowadays there are several motifs whose regulator is yet to be known.

The abundance of computationally derived sequence motifs is crucial in the actual definition of genetic regulatory networks and in deciphering the regulatory program of individual genes, which, summing up; make them very important tools in the post-genomic era of computational biology.

In the field of pattern recognition, Hidden Markov models (HMMs) are popular tools, and have been developed in genomic sequence analysis.

In a motif finding problem we can observe the nucleotides “A”, “G”, “C”, “T” following a specific order and having a variable distance between the different number of motifs. The probabilities of observing each nucleotide at each position are given by positional weight matrices (PWM).

The HMMs models are based in hidden states; that in the case of motif finding are the background state (nucleotides that do not participate in the motif), and the motif states; from

the first to the last position participating in the motif. The probability of observing letters from each state is called “emission probability”, in the case of the observations of the motif states are given by the PWM; and in the case of observations of the background state, as they can be purely random, the emission probabilities will be uniform (equal to 0.25).

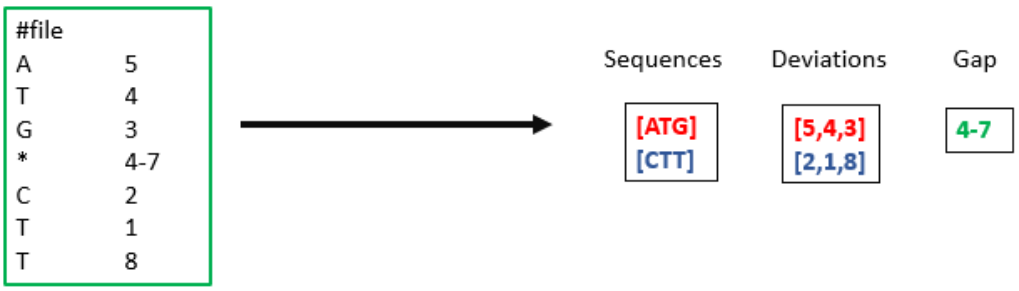
There are different algorithms based on HMM to find motifs in a given segment of sequence, but all of them require examples of the motif to work (training data), as the emission probabilities are estimated by observed frequencies (2).

In our project, we are exploring a different method which is based on an approximately similar approach; the training and its resultant positional weight matrix is “summed up” in a weighting file that adds a deviation when the found nucleotide is different from the one that would be found in an ideal consensus sequence (highest emission probability at a specific location in the PWM). Thus, even though we are not considering the probabilities that every nucleotide would have, we are taking into account their appearance and permitting its presence, setting a deviation threshold. The size of this deviation threshold will determine the flexibility of the algorithm to look deeply into the appearance of nucleotides with lower emission probabilities.

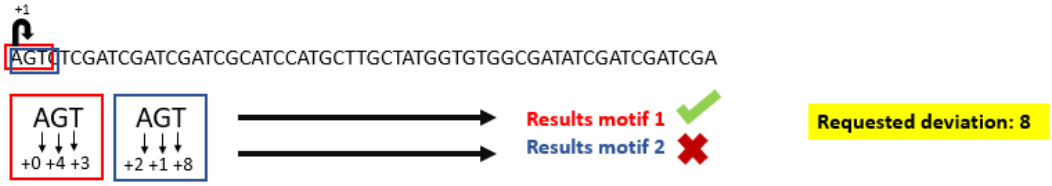
ALGORITHM DESIGN

The algorithm of the program is the following:

1. First, it saves the sequences of the different parts of the motif (a motif can be formed by different parts separated by gaps of a variable size), their length and the gap existing between them. Also, it saves the weighted deviation if the consensus nucleotide does not appear at the correct position. All this information must be given in a separate file.

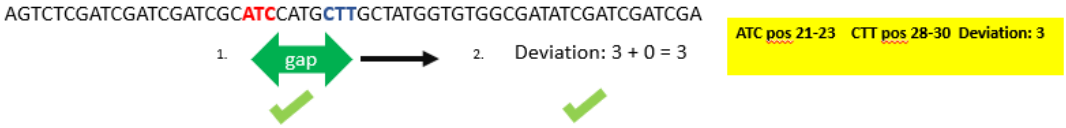


2. Afterwards, it runs through the sequences of a file looking for the different motifs saved beforehand. It will look if the motif is present starting in each nucleotide of the sequence, if one of the nucleotides is not in the consensus position of the motif, it adds its deviation to a variable. If the total deviation is smaller than the threshold set up, it will save the starting and ending position, as well as the calculated deviation as a match.



3. If the motif is formed by two parts or more parts, it will look at the position of each match found. If the position of each indicates that they are separated by a gap of a correct size (determined in the first step), it will add the deviation calculated for each of

them. Then, if the addition of the deviations is still smaller than the threshold set up, it will save the sequences and the total deviation as a new result. To do so, in case there are more than two parts; it will sew the different found parts, looking at one gap at a time.



PROGRAM DESIGN

FUNCTIONS

The design of the program is mainly based on four functions:

1. The motif creator

In this function, an input file containing the motif the user is searching for, is used in order to identify the basic variables of the program. According to the character each line starts with, the line is identified either as a comment, as part of a motif or as an interval of unimportant positions between parts, and it is appended in the appropriate variable. These variables are:

- The **specific motif** the program should search for, saved in a list of lists. Each inner list represents a part of the motif and each part is described using lists which contain the expected nucleotide in the first position and the score of a mismatch in the second position.
- The **length of the gap** between the different parts of the model
- The **size** (length) of each part of a motif.

2. The match finder

This function uses as input a given sequence, a specific motif that was identified earlier using the motif creator function, it's size and a threshold for deviation. Starting from the first nucleotide, it compares a part of the sequence with the specific motif. For every mismatch between the nucleotides, it adds the penalty of the specific position in a variable. When the length of the sequence being checked is equal to the size of the motif, the deviation is compared to the threshold.

If the deviation is lower than the threshold, the match is appended to a list of lists. Each outer list represents a match and each inner list consists of the start position, the end position and the calculated deviation of the specific match. Moving on, the following nucleotide is used as the start of the sequence being checked, the variable is restarted and the deviation is calculated in the same way. The same procedure is repeated for the whole sequence, and when everything has been checked the **list of all successful matches** is returned from the function.

3. The deviation func and total deviation

These two functions are used in order to identify matches in the sequence when the specific motif contains more than one part. The matches identified for each part using match finder, their size and the deviation are given as input.

In case the motif contains two parts, only deviation func is used. Every possible combination of matches from each part is examined. In order to be successful, a match should meet all of the following criteria.

- The match of the second part should start after the end of the match of first part
- The difference between the end of the first part and the start of the second, should be on a specific interval specified using motifs creator (gap variable)
- The sum of deviations of the two matches should be smaller than the threshold

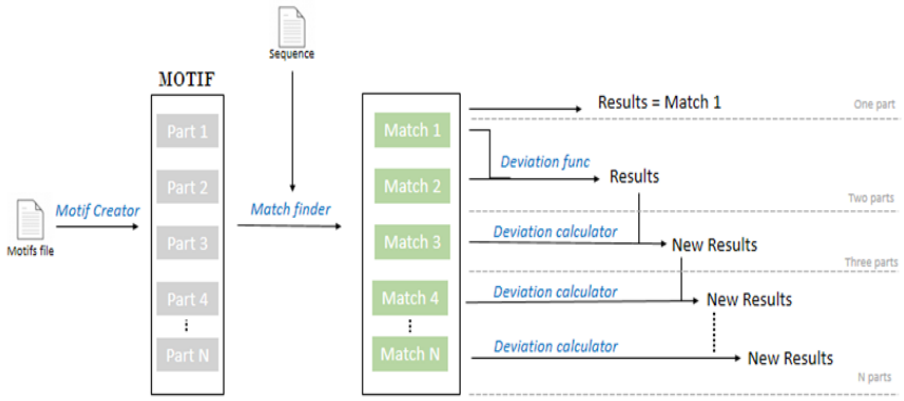
All successful results are appended in a list of lists, which is returned to the main program. Each outer list represents a match and each inner list saves the position in which the match starts, the position where it ends, the total deviation and the difference between the two parts.

When there are more than two parts in the motif, the deviation func is used to identify matches between the first two parts as described earlier. Following that, the total deviation function is used. The logic behind this function and deviation func is the same, the only difference is that now, the analyzed combinations are those of the successful results between the first two parts and the matches of the third part. Each pair has to fulfill the same criteria as the ones of deviation func in order to be successful.

The same procedure is repeated for all the parts of motifs, where the results of the previous parts are each time combined with the following part. The results are saved in the same way, a list of lists. Each outer list represents a match, and the inner list consists of start position, end position, total deviation, and a list of the differences between all parts of the motif. When all the combinations have been examined, the **list of successful results** is returned to the program.

MAIN PROGRAM

The program works in the following way:



Main Program design. Functions are highlighted in blue

- ➔ The necessary information is extracted from the input motifs file using motif creator function.
- ➔ The file containing sequences is opened and each line is checked.
  - ➢ If the line contains a header, it is saved and written in an output file, then it moves on to the next line and starts saving the sequence in a variable, till the next time it meets a header line.
  - ➢ When it meets the next header line it saves it, then the sequence already saved in the variable is analyzed using the match finder function. According to the number of parts in the motif:

    If the motif has only one part, the parts of the sequence that are a match are written in the output file.

    If there are two parts, deviation func is used in order to identify successful matches. Then results are written in the output file.

    If there are more than two parts, the deviation func is used for matches between the first two parts and then the deviation calculator for the rest of the parts. Final results are written in the output file.

  - ➢ The header line saved is printed out in the file and the variables are restarted in order to parse for the following sequence. The same procedure is repeated for all the sequences till the last header is met. After that the sequence is saved and analyzed as before one last time.
- ➔ Files are closed and results are ready and saved in a file of name indicated in the screen

PRINTING OF RESULTS

A function called position finder is used to help print out the results. This function takes as input the list of results retrieved earlier using deviation func or total deviation, the position of a specific result on that list and which part of the motif it reassembles. Then it calculates the start and end position in the sequence, and the specific nucleotides in these positions and returns them in the main program to be written in the output file.

When the motif contains more than two parts, the function is only used to calculate the position of the first part. Then the length of the gap that is saved in the results list is added in the end position, in order to get the start position of the next part. Also, the size of each part of motifs is known from the motifs creator function, so the end position can also be calculated. The same procedure is repeated for every part of a result.

After each match the calculated total deviation is also printed, it moves on to the next result on the list and repeats the same till all of the results are written in the file.

PROGRAM MANUAL

FILES NEEDED

- Sequences´ file: this file contains the sequences in which we are going to look for the given motifs. It must contain sequences in fasta format, this means that the identification of the sequence should be in a different line and preceded by a “>” symbol. Then the sequence should start at the next line. Example:

```
>gnl|ti|1270769348 name:AWGF1001.y1 mate:1270769347
AAAAAAAAAAAGAGAAAAACAACCAAGGAAGGAACCNNCGTATAGCCGGTCGGGTACGGATCATCGGAGA
TTGCGACAGATCTTCGGCTCCGTTGACAAAGCCATAGCGGCATTCAAATGAACGAACGCCGTTACGAAGT
GATCGGTGTGGTGAGGATGGAAAATATCAACCCTCACCGAAGATCGCAGTTCGCACGTCCTGCCGATCCT
TCAGGCACCCACAAGTCCACTACGCTCGTCGTTCGCACCACGCGCGATCCCGCGCAGTCGGCCCCTGAAG
TCGACTACACCCTTCGCAAGCTCGATAGCAGGCTTCCGTTCGTGATCAAAACGTGGCCGCAACAACTTGA
CAGCGCCCTTTTCGCGTCGCGTGTCGCTACGGTCGCGCTCGGTATTCTGGGCGCACTATGATCTATATCA
GCGATCTCTGGGGTATGGCGTACGATGTCGTATGGAGACTCATGCGTGATCTGTGAATCGGCGTTGCTCT
TGGCGCCCAGAGCACGCAGGACGCATGTGCCGTGCTCCGTACTCGATTGAGCCTTGTCCCTATCGGCTAG
CTGATCGGACTTGTGCTATGATTCAGCATACGCGGCGTTCTTTCGCTATGTGGTATACTCGATGTAACAG
CGAAGCGATACGCTAGTCCTTGGTGCTGTGGTGTGCACGCGTGTACCAGGTGGGCCTGGTCGCTGCTTGG
ATCCTTGATCGGTGCGCGCTTCGCTCTTAGCCCTGACACGTGGTCCCGTGATCCATAAACCGGTTCCCGG
TTCCCATCTTCTCTCGACTCTTAACTCTGGTAGACGTGCGGATATCAGATATATCGTGAATCGAGTAGGA
AAATGGAGAAAGGGAATGGGAAGGCGGAATGCGTTAATCAGTTAATAGTACATTTCAGGAATAAGGGAAG
ATGCTTAATATCTAGTCGCATGCCCGATCGAAAATCGTGGCCACGCGTACGGTGGTCTCTTCGTGTCATG
CTCGTACCGCGAACCTCGACATCGGTCAGCCGTTGACATGCAGAAGCTTCGACGTTCATCCACAGCCAGG
GCGTGGGTTTTGTGTTGCTTGCGCGGGCGCTGAATTCCAGATGCCGGTTACTTCTGCATATGCCTTCAAC
TTCACTGAACCTAAGGGAAGCCGAGTGCGGAAGGGCAACCTTTCCATTGCGGAACGTTAAAGTTGTCCTT
TCCCTGGTTGNNCC
```

- Motifs´ file: this file contains the motif that we are going to look into the sequence. The format is the following one:

    - The lines that start by a “#” are comments.

    - The lines that start by a “*” character indicate gaps between parts of the motif. The gaps can be undetermined (interval of numbers of nucleotides) or determined (exact number of nucleotides). The size of the gap is indicated after a tab space.

    - The lines that do not start with any of the previous symbols indicate motif sequence (in case of genomic searching, they will start with a “T”, an “A”, a “C” or a “G”). The line will start with the nucleotide that appears in the consensus motif followed by a tab space, and then; the deviation value that will be added if it does not appear in the sequence given. Example:

```
# -35 element
T       7
T       8
G       6
A       5
C       5
A       5
# intervening unimportant bases
*       15-21
# -10 element
T       8
A       8
T       6
A       6
AT      5
T       8
```

INPUT

To use the program, you can use two different methods:

1. Open only the program. After doing so, the program will ask for the file with the sequences in fasta format, for the file that contains the motif and the deviation that will be set up as the deviation threshold for acceptance of motifs.

```
javihersan@LAPTOP-PSBH70NI:~/Pythoncourse$ ./final1.py
Please enter sequences file: motif.fsa.txt
Please enter motifs file: motifs.txt
Which deviation should I accept? 20
Your results are ready in file: found_motifs.txt
```

2. Open the program, the file containing the sequences (they must be in fasta format), the file containing the motifs and the deviation that you want to set as the deviation threshold (this order must be kept).

```
javihersan@LAPTOP-PSBH70NI:~/Pythoncourse$ ./final.py motif.fsa.txt motifs.txt 25
Your results are ready in file: found_motifs.txt
```

OUTPUT

If the correct input is given, the output is given in a text file format, this file is called "found_motifs.txt". The file possesses the following structure:

```
>gnl|ti|1270769347 name:AWGF1001.x1 mate:1270769348
1. Pos: 930-936  Sequence: CTGACG      2. Pos: 963-969  Sequence: CTGTGC      Deviation = 18
1. Pos: 1045-1051  Sequence: TTGTGA    2. Pos: 1072-1078  Sequence: AAGAGT    Deviation = 24
1. Pos: 1078-1084  Sequence: CTGAGA    2. Pos: 1109-1115  Sequence: CGCGGA    Deviation = 23
```

- First it will indicate the fasta identification of the sequence in which it has found the motif (the sequences file will normally include several fasta sequences).

- Afterwards, it will show the different motifs found with a deviation lower than the threshold given. It will write one motif per line.

- The motif lines will include the following information: position of the parts, sequence of the parts; and total deviation of the found motif.

If the input is not correctly given, the program will indicate the errors that could have been committed: error in the order of files, incorrect files, etc.

EXAMPLES

INCORRECT INPUTS:

Incorrect structure of file:

```
# -35 element
T       7
T       8
G       K
A       5
C       5
A       5
# intervening unimportant bases
*       15-21
# -10 element
T       8
A       8
T       6
A       6
AT      5
T       8
```

```
javihersan@LAPTOP-PSBH70NI:~/Pythoncourse$ ./final.py
Which deviation should I accept? 25
Penalty is not defined correclty in motifs file. Please check your input
```

Incorrect files or order of files given:

```
javihersan@LAPTOP-PSBH70NI:~/Pythoncourse$ ./final.py motifs.txt motif.fsa.txt 25
Please check motifs file
```

Incorrect number of arguments given:

```
ellide@LAPTOP-R73GJRQ2:~/22110/Project$ ./final_project.py motifs.txt 25
./program.py <fasta file> <motifs file> <deviation>
```

CORRECT INPUTS:

- **1 PART MOTIF (Deviation requested = 10)**

```
# -35 element
T       7
T       8
G       7
A       5
C       5
A       5
```

```
>gnl|ti|1270769348 name:AWGF1001.y1 mate:1270769347
Pos: 70-76,  Sequence: TTGCGA   Deviation = 10
Pos: 92-98,  Sequence: TTGACA   Deviation = 0
Pos: 346-352,  Sequence: TTGACA Deviation = 0
Pos: 489-495,  Sequence: TTGGCG Deviation = 10
Pos: 536-542,  Sequence: TTGAGC Deviation = 10
Pos: 543-549,  Sequence: TTGTCC Deviation = 10
Pos: 704-710,  Sequence: TTGATC Deviation = 10
Pos: 732-738,  Sequence: CTGACA Deviation = 7
```

- **2 PARTS MOTIF (Deviation requested = 25)**

```
# -35 element
T       7
T       8
G       6
A       5
C       5
A       5
# intervening unimportant bases
*       15-21
# -10 element
T       8
A       8
T       6
A       6
AT      5
T       8
```

```
>gnl|ti|1270769347 name:AWGF1001.x1 mate:1270769348
1. Pos: 930-936  Sequence: CTGACG       2. Pos: 963-969  Sequence: CTGTGC       Deviation = 18
1. Pos: 1045-1051  Sequence: TTGTGA     2. Pos: 1072-1078  Sequence: AAGAGT     Deviation = 24
1. Pos: 1078-1084  Sequence: CTGAGA     2. Pos: 1109-1115  Sequence: CGCGGA     Deviation = 23
```

- **>2 PARTS MOTIF (Deviation requested = 30)**

```
# -35 element
T       7
T       8
G       6
A       5
C       5
A       5
# intervening unimportant bases
*       15-21
# -10 element
T       8
A       8
T       6
A       6
AT      5
T       8
# intervening unimportant bases
*       10-15
# New element
G       4
C       6
A       5
```

```
>gnl|ti|1270769354 name:AWGF1005.y1 mate:1270769353
1. Pos: 55-61  Sequence: TTGGCT,        2. Pos: 79-85  Sequence: GAAAAT,        3. Pos: 99-102  Sequence: GCA,  Deviation = 24
1. Pos: 121-127  Sequence: TCCATA,      2. Pos: 144-150  Sequence: TAGAAT,      3. Pos: 165-168  Sequence: CCA, Deviation = 29
1. Pos: 612-618  Sequence: GTGAAA,      2. Pos: 639-645  Sequence: GATTAT,      3. Pos: 659-662  Sequence: GCA, Deviation = 26
1. Pos: 618-624  Sequence: TTCAAT,      2. Pos: 639-645  Sequence: GATTAT,      3. Pos: 659-662  Sequence: GCA, Deviation = 30
1. Pos: 999-1005  Sequence: TTGTCA,     2. Pos: 1021-1027  Sequence: CAGCAT,    3. Pos: 1037-1040  Sequence: GCC,       Deviation = 30
1. Pos: 999-1005  Sequence: TTGTCA,     2. Pos: 1021-1027  Sequence: CAGCAT,    3. Pos: 1038-1041  Sequence: CCA,       Deviation = 29
1. Pos: 999-1005  Sequence: TTGTCA,     2. Pos: 1021-1027  Sequence: CAGCAT,    3. Pos: 1042-1045  Sequence: GCT,       Deviation = 30
```

In the case in which the program is not able to find a motif in the whole sequence, the output file will write the following statement:

```
>gnl|ti|1270769355 name:AWGF1006.x1 mate:1270769356
No results found
```

RUNTIME ANALYSIS

The runtime analysis in Big O notation, which denotes the complexity of the program; is the following one:

$$O(l + k \cdot n^m + k \cdot n(log\ n))$$

- The program runs and saves l lines given in the sequence file.
- For each nucleotide (n) loops m times, which is the size of each motif; it will do so k times, which is a variable that indicates the number of parts that the motif we are looking for has.
- After that it will look for the gaps existing between the matches found, the number of matches correlates with the number of nucleotides n (more nucleotides more matches) in a logarithmic way, as it will discard those matches that do not comply with the gap size statement. The gap finding also depends on the number of parts of our motif (k).

As the complexity of the program is dependent on the more complex part of the algorithm, the Big O of the program will be:

$$\boldsymbol{O(k \cdot n^m)}$$

In terms of time, when looking for a standard -35 element / -10 element promoter, with a given deviation of 15 in a file containing 50 sequences. The program takes approximately one second (1,408 seconds) to compute a result.

CONCLUSION

The program performs a correct searching of motifs into sequences, being flexible to the size of the motifs to be found and to the strictness required to match the consensus sequence.

In the process of coding, it has been considered to address most of the possible errors that could appear at the time of computing, thus; they have been handled and different errors would jump in variable situations to achieve an optimal operation and be managed by the user.

The code has been thought and designed in order to be capable of handling searching in different fields (not only life science), as long as the sequences given follow a fasta format of identification. Because of this characteristic of our code, the quality of the results also relies on the quality of the input sequence provided by the user. Since the code is designed to be

universal, there is no input control for the sequences file, thus the program trusts the user that the input file will contain header lines starting with “>” followed by the correct sequence.

Despite the achievements and the utility that the designing of the program implies, the program possesses some weakness as it could be a too strict control of the structure of the input files, as these need to have a too determined structure to be handled. Also, to be more optimal in terms of looking for sequences in genome data, it would be nice to add different deviations depending on the nucleotide that appears (if they are different from the one in the consensus sequence). Thus, it would not need training data as HMM models or machine learning ones; but it would take into account the emission probabilities mentioned in the theory part; as they are not considered in our current program.

REFERENCES

1. D’Haeseleer, P. What are DNA sequence motifs? *Nat. Biotechnol.* **24**, 423–425 (2006).
2. Wu, J. & Xie, J. Hidden Markov Model and Its Applications in Motif Findings. *Methods Mol. Biol.* **620**, 405–416 (2010).