

Implementação

A implementação para encontrar motivos em sequências de DNA não alinhadas foi baseada no algoritmo proposto no artigo Hertz, G.Z. and Hartzell, G.W. and Stormo, G.D. Identification of consensus patterns in unaligned DNA sequences known to be functionally related. Comput Appl Biosci.1990, Vol. 6 (2).

A função principal começa construindo o primeiro conjunto de matrizes relativo à primeira sequência considerada. Em seguida, para cada sequência, atualiza o conjunto de matrizes e, por fim, seleciona a melhor dentre elas com base no cálculo do termo *I* (*information content*).

```
40 def find_motifs(sequences, motif_len):
41
42     matrices = build_first_matrices(sequences[0], motif_len)
43     for sequence in sequences[1:]:
44         matrices = update_matrices(sequence, matrices, motif_len)
45
46     best_matrix = select_matrix(matrices)
47     return best_matrix
```

Nesta implementação, as matrizes são representadas por uma tripla. O primeiro elemento da tripla corresponde a uma lista contendo todas as sequências consideradas para aquela matriz em questão, que é o segundo elemento da tripla. Por fim, terceiro elemento é o valor *I* da matriz, definido no artigo propôs o método.

A função que constrói as matrizes iniciais recebe como parâmetros a sequência inicialmente considerada e o tamanho definido do motivo. O número de matrizes a ser considerado (tanto inicialmente, quanto nas iterações do algoritmo principal) é dado pelo tamanho da sequência subtraído pelo tamanho do motivo mais um. E, tendo esta quantidade, o algoritmo itera construindo uma matriz para cada *l*-mer possível.

```
52 def build_first_matrices(sequence, motif_len):
53
54     matrices = []
55     num_matrices = len(sequence) - motif_len + 1
56
57     for i in range(num_matrices):
58         l_mer = get_l_mer_by_iteration(sequence, motif_len, i)
59         matrix = build_matrix([l_mer])
60         matrices.append(matrix)
61
62     return matrices
```

A função que constrói matrizes também é utilizada na função que atualiza as matrizes do loop principal (em find_motif). Esta função considera uma lista de l-mers e constrói as matrizes considerando que todas terão quatro linhas (referente às quatro bases) e |l-mer| colunas (o tamanho dos l-mers). Cada elemento da matriz representa o número de “matches” naquela posição do l-mer para aquela base.

```

73 def build_matrix(l_mers):
74
75     num_cols = len(l_mers[0])
76     matrix = []
77
78     for base in ['a', 'c', 'g', 't']:
79         row = []
80         for position in range(num_cols):
81             matches = 0
82             for l_mer in l_mers:
83                 if l_mer[position] == base:
84                     matches += 1
85             row.append(matches)
86         matrix.append(row)
87
88     I = compute_I(matrix, l_mers)
89
90     return (l_mers, matrix, I)

```

A implementação da função que computa o I foi baseada na Fórmula (1) do artigo:

$$I = \sum_{i=1}^L \sum_{b=A}^T \frac{N_{bi}}{N} \log_2 \frac{N_{bi}/N}{P_b}$$

```

93 def compute_I(matrix, l_mers):
94
95     num_sequences = len(l_mers)
96     pa = get_genomic_frequency(l_mers, 'a')
97     pc = get_genomic_frequency(l_mers, 'c')
98     pg = get_genomic_frequency(l_mers, 'g')
99     pt = get_genomic_frequency(l_mers, 't')
100     frequencies = [pa, pc, pg, pt]
101
102     I = 0
103
104     for idx, row in enumerate(matrix):
105         for col in row:
106             N_bi = col
107             if N_bi == 0:
108                 continue
109
110             I += (N_bi / num_sequences) * \
111                 log2((N_bi / num_sequences) / frequencies[idx])
112
113     return I

```

Primeiramente esta função calcula as “frequências genômicas” de cada base, dividindo o número de vezes que a base aparece nas sequências consideradas pelo número de bases no total (das sequências consideradas). Este cálculo foi inferido pois não fica claro no artigo o que seria o *genomic frequency*. Após este procedimento, para cada elemento da matriz **diferente de 0** (pois $\log_2 0$ não está definido e resultaria num 0 multiplicando $-\infty$), soma-se ao 1 o \log_2 de N_{bi} / N dividido pela frequência da base considerada, tudo isso multiplicado pelo termo N_{bi} / N . N_{bi} é definido como o número de matches do elemento em questão da matriz, e N , o número de sequências indexadas por aquela matriz.

Nota: eu investiguei a corretude do cálculo desse termo I para a matriz E de exemplo na Figura 1. Segundo o artigo, a matriz possui um I de 6.1, mas encontrei um valor de 5.92 (que acredito ser originado dos arredondamentos intermediários e falta de manipulações algébricas), segue a matriz E e o cálculo feito no Google Colaboratory.

E

	A	C	T	G	A	A
	A	G	C	G	T	C
	C	T	T	G	C	C
A	2	0	0	0	1	1
C	1	1	1	0	1	2
G	0	1	0	3	0	0
T	0	1	2	0	1	0

$I = 6.1$

```
[3] PA = 4/18
    PC = 6/18
    PG = 4/18
    PT = 4/18

(2/3) * log2((2/3) / PA) + \
(1/3) * log2((1/3) / PA) + \
(1/3) * log2((1/3) / PA) + \
(1/3) * log2((1/3) / PC) + \
(1/3) * log2((1/3) / PC) + \
(1/3) * log2((1/3) / PC) + \
(1/3) * log2((1/3) / PC) + \
(2/3) * log2((2/3) / PC) + \
(1/3) * log2((1/3) / PG) + \
(3/3) * log2((3/3) / PG) + \
(1/3) * log2((1/3) / PT) + \
(1/3) * log2((1/3) / PT) + \
(2/3) * log2((2/3) / PT)
```

5.92481250360578

Após a construção das primeiras matrizes em `find_motif`, a cada iteração, uma nova sequência é considerada, são criadas aquele número de matrizes (uma para cada l-mer possível da nova sequência) para cada matriz inicial, mas (e eis o diferencial do algoritmo proposto), ao calcular o termo I daquelas matrizes, só permanece, para cada matriz inicial, uma única matriz (a com maior I). É exatamente isso que a função `select_matrix` se propõe a fazer, simplesmente retornar a melhor dentre uma lista de matrizes (é utilizada tanto em `update_matrix`, quanto em `find_motif` para retornar o resultado final).

```

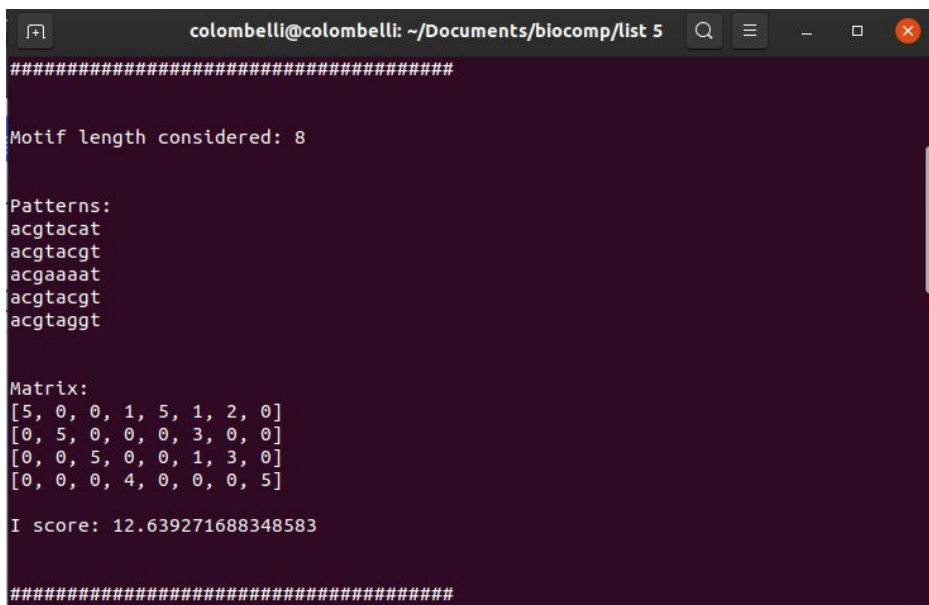
127 def update_matrices(sequence, matrices, motif_len):
128
129     # For each matrix
130     num_matrices = len(matrices)
131     updated_matrices = []
132
133     for matrix in matrices:
134         current_matrices = []
135
136         for i in range(num_matrices):
137             l_mers = deepcopy(matrix[0])
138             new_l_mer = get_l_mer_by_iteration(sequence, motif_len, i)
139             l_mers.append(new_l_mer)
140             matrix_temp = build_matrix(l_mers)
141             current_matrices.append(matrix_temp)
142
143         best_matrix = select_matrix(current_matrices)
144         updated_matrices.append(best_matrix)
145
146     return updated_matrices

```

Resultados

Primeiro conjunto de teste

Motivo de tamanho 8 aceitando 2 mutações



```

colombelli@colombelli: ~/Documents/biocomp/list 5
#####
Motif length considered: 8

Patterns:
acgtacat
acgtacgt
acgaaaat
acgtacgt
acgtaggt

Matrix:
[5, 0, 0, 1, 5, 1, 2, 0]
[0, 5, 0, 0, 0, 3, 0, 0]
[0, 0, 5, 0, 0, 1, 3, 0]
[0, 0, 0, 4, 0, 0, 0, 5]

I score: 12.639271688348583

#####

```

Se considerarmos arbitrariamente a primeira sequência como baseline, podemos ver que as mutações realizadas em cada l-mer podem ser destacadas da seguinte maneira:

```
colombelli@colombelli: ~/Documents/biocomp/list 5

#####

Motif length considered: 8

Patterns:
acgtacat
acgtacgt
acgaaaat
acgtacgt
acgtagggt

Matrix:
[5, 0, 0, 1, 5, 1, 2, 0]
[0, 5, 0, 0, 0, 3, 0, 0]
[0, 0, 5, 0, 0, 1, 3, 0]
[0, 0, 0, 4, 0, 0, 0, 5]

I score: 12.639271688348583

#####
```

Como em cada sequência o motivo encontrado só realiza uma ou duas mutações (por mais que em lugares diferentes), os resultados foram considerados satisfatórios.

Podemos, também, utilizar a abordagem descrita no artigo e analisar a matriz resultante definindo nosso baseline da seguinte maneira: para cada posição, considera-se a base original como aquela que mais teve matches na matriz para aquela posição. Assim, e considerando que as linhas da matriz estão organizadas para representarem cada base em ordem alfabética, temos como baseline:

A C G T A C G T

E, com isso, podemos plotar as seguintes mutações em cada motivo de cada sequência:

1. Posição 7 (G-A)
2. Nenhuma
3. Posição 4 (T-A), posição 6 (C-A) e posição 7 (G-A)
4. Nenhuma
5. Posição 6 (C-A)

E assim, se concluiria que o algoritmo falhou em encontrar um motivo de tamanho 8 com apenas duas mutações, pois o motivo da sequência 3 realiza três mutações.

Motivo de tamanho 5 aceitando 3 mutações

Para o motivo de tamanho 5, utilizando as duas abordagens para definição do baseline, o mesmo motivo seria encontrado: C G T A C

E, assim, apenas a terceira sequência conteria uma única mutação: posição 4 (A-G). Seguem os resultados:

```
colombelli@colombelli: ~/Documents/biocomp/list 5
#####
Motif length considered: 5

Patterns:
cgtac
cgtac
cgtgc
cgtac
cgtac

Matrix:
[0, 0, 0, 4, 0]
[5, 0, 0, 0, 5]
[0, 5, 0, 1, 0]
[0, 0, 5, 0, 0]

I score: 8.829613568458788
#####
```

Motivo de tamanho 3 aceitando 1 mutação

Fato semelhante ao anterior ocorre aqui, porém, a mutação agora é na quarta sequência, na primeira posição (C-A).

```
colombelli@colombelli: ~/Documents/biocomp/list 5
#####
Motif length considered: 3

Patterns:
ctg
ctg
ctg
atg
ctg

Matrix:
[1, 0, 0]
[4, 0, 0]
[0, 0, 5]
[0, 5, 0]

I score: 4.754887502163468
#####
```

Segundo conjunto de teste

Motivo de tamanho 3 aceitando 1 mutação

Agora, a primeira abordagem resulta no baseline GTC (com 6 sequências sofrendo uma mutação na primeira posição), e a segunda abordagem resulta no baseline ATC (com 4 sequências sofrendo uma mutação na primeira posição).

Seguem os resultados:


```
colombelli@colombelli: ~/Documents/biocomp/list 5

Motif length considered: 3

Patterns:
gtc
atc
gtc
atc
atc
atc
atc
gtc
gtc
atc

Matrix:
[6, 0, 0]
[0, 0, 10]
[4, 0, 0]
[0, 10, 0]

I score: 4.754887502163468
```

Motivo de tamanho 5 aceitando 2 mutações

```
colombelli@colombelli: ~/Documents/biocomp/list 5

Motif length considered: 5

Patterns:
tacgg
tactg
aactg
tacgg
tactg
tactc
tactg
tacgg
tactg
tactg

Matrix:
[1, 10, 0, 0, 0]
[0, 0, 10, 0, 1]
[0, 0, 0, 3, 9]
[9, 0, 0, 7, 0]

I score: 8.087294300596927
```

Baseline da primeira abordagem: T A C G G

Mutações (por sequência):

2. Quarta posição (G-T)
3. Primeira posição (T-A) e quarta posição (G-T)
4. Nenhuma
5. Quarta posição (G-T)
6. Quarta posição (G-T) e quinta posição (G-C)
7. Quarta posição (G-T)
8. Nenhuma

9. Quarta posição (G-T)
10. Quarta posição (G-T)

Baseline da segunda abordagem: T A C T G

Mutações (por sequência):

1. Quarta posição (T-G)
2. Nenhuma
3. Primeira posição (T-A)
4. Quarta posição (T-G)
5. Nenhuma
6. Quinta posição (G-C)
7. Nenhuma
8. Quarta posição (T-G)
9. Nenhuma
10. Nenhuma

As duas abordagens resultaram ou em no máximo duas mutações em uma única sequência, portanto foram considerados e considerou-se satisfatório o desempenho do algoritmo.