

# SemiGlobalAlignment

Frank Li

April 3, 2020

## 1 Usage of the script

The code is designed for dealing with overlap alignment(aka semi-global alignment). For example, If we want to check if Sequence1(ATAGGTGATATA) will partially align with Sequence2(ATATACTGG), we just need to run following code in your terminal:

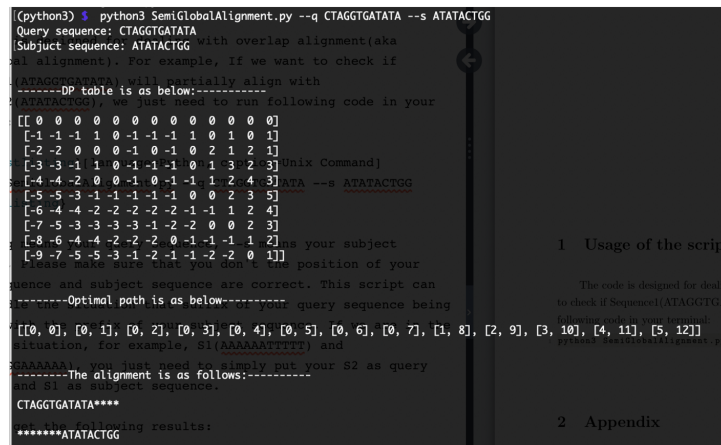
```
1 python3 SemiGlobalAlignment.py --q CTAGGTGATATA --s ATATACTGG
```

Listing 1: Unix Command

Where `-q` means your query sequence, `-s` means your subject sequence, Please make sure that the position of your query sequence and subject sequence are correct. This script can only handle the situation that suffix of your query sequence being aligned with the prefix of your subject sequence. If we are in the opposite situation, for example, S1(AAAAAATTTTT) and S2(GGGGGGAAAAAA), you just need to simply put your S2 as query sequence and S1 as subject sequence.

**Please install getopt package and numpy package.**

You will get the following results as [Figure 1](#):



```
(python3) 5 python3 SemiGlobalAlignment.py --q CTAGGTGATATA --s ATATACTGG
Query sequence: CTAGGTGATATA
Subject sequence: ATATACTGG with overlap alignment(aka
partial alignment). For example, If we want to check if
Sequence1(ATAGGTGATATA) will partially align with
Sequence2(ATATACTGG), we just need to run following code in your
terminal:
python3 SemiGlobalAlignment.py --q CTAGGTGATATA --s ATATACTGG
-----DP table is as below-----
[[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [-1 -1 -1 0 -1 -1 -1 1 0 1 0 1]
 [-2 -2 0 0 -1 0 -1 0 2 1 2 1]
 [-3 -3 -1 1 0 -1 -1 0 1 3 2 3]
 [-4 -4 -2 0 0 -1 0 -1 -1 1 2 4 3]
 [-5 -5 -3 -1 -1 -1 -1 0 0 2 3 5]
 [-6 -4 -4 -2 -2 -2 -2 -1 1 2 4]
 [-7 -5 -3 -3 -3 -1 -2 -2 0 2 3]
 [-8 -6 -4 -4 -2 -2 -2 0 -1 -1 1 2]
 [-9 -7 -5 -5 -3 -1 -2 -1 -2 0 1]]
-----Optimal path is as below-----
[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [1, 8], [2, 9], [3, 10], [4, 11], [5, 12]]
The alignment is as follows:
CTAGGTGATATA****
*****ATATACTGG
```

Figure 1: DP table, optimal path and final alignment

## 2 Appendix

```
1 import os
2 import sys
3 import getopt
4 import numpy as np
5
6 def DP(query,subject,match = 1, mismatch = -1, indel = -1):
7     n = len(subject) + 1
8     m = len(query) + 1
9     dp_table = np.zeros((n,m),dtype=np.int16) #int16 means each number at most occupy two bytes,
10     ranging from -32768 ~ 32767
11     #print(dp_table)
12     # initilize
13     dp_table[0,:] = 0
14     for i in range(n):
15         dp_table[i,0] = 0 + mismatch*i
16     #print(dp_table)
17     # forward calculation
18     for i in range(1,n): # row
19         for j in range(1,m): # column
20             if query[j-1] == subject[i-1]:
21                 diag_move = dp_table[i-1,j-1] + match
22             else:
23                 diag_move = dp_table[i-1,j-1] + mismatch
24             hor_move = dp_table[i,j-1] + indel
25             ver_move = dp_table[i-1,j] + indel
26             dp_table[i,j] = max(diag_move,hor_move,ver_move)
27
28     print('\n')
29     print('-----DP table is as below:-----\n')
30     print(dp_table)
31     # backward tracking
32     row, column = 0,0
33     trace = []
34     max_last_column_index = np.argmax(dp_table[:,m-1])
35     #print(max_last_column_index)
36     row = max_last_column_index
37     column = m-1
38     trace.append([row,column])
39     #print(row,column)
40     current_entry = dp_table[row,column]
41     #print(current_entry)
42     while not (row == 0 and column ==0):
43         if row > 0 and column > 0:
44             if current_entry == dp_table[row-1,column-1] + 1 or current_entry == dp_table[row-1,
45             column-1] - 1: # diag
46                 row,column = row-1,column-1
47                 trace.append([row,column])
48                 current_entry = dp_table[row,column]
49             elif current_entry == dp_table[row-1,column] -1: # vertical
50                 row,column = row-1, column
51                 trace.append([row,column])
52                 current_entry = dp_table[row,column]
53             elif current_entry == dp_table[row,column-1] - 1: # horizontal
54                 row,column = row, column-1
```

```

52         trace.append([row,column])
53         current_entry = dp_table[row,column]
54         elif row == 0 and column > 0: # must horizontal
55             row,column = row, column-1
56             trace.append([row,column])
57             current_entry = dp_table[row,column]
58         elif row > 0 and column == 0: # must vertical
59             row,column = row-1, column
60             trace.append([row,column])
61             current_entry = dp_table[row,column]
62     trace = trace[::-1] # reverse the trace
63     print('\n')
64     print('-----Optimal path is as below-----\n')
65     print(trace)
66     # format the alignment
67     for m in range(len(trace)):
68         if not trace[m][0] == 0:
69             break
70     match_pos = m-1 # matching starts index 2 (third letter) as below
71     leading_star_num = match_pos # fill up 2 * at the beginning of subject sequence
72     trailing_star_num = leading_star_num + len(subject) - len(query) # fill up 2 * at the end of
73     query sequence
74     print('\n')
75     print('-----The alignment is as follows:-----\n')
76     print('{0:*<{1}s}\n'.format(query,len(query)+trailing_star_num))
77     print('{0:*>{1}s}\n'.format(subject,len(subject)+leading_star_num))
78 if __name__ == "__main__":
79     # run as python3 SemiGlobalAlignment.py -q ATT -s TTT
80     try:
81         options, remainder = getopt.getopt(sys.argv[1:], 'hq:s:', ['help', 'q=', 's=']) # getopt usage
82         refer to pymotw.com/3/getopt/
83     except getopt.GetoptError as err:
84         print('ERROR:', err)
85         usage()
86         sys.exit(1)
87     for opt, arg in options:
88         if opt in ('-q', '-q'):
89             query = arg
90             print('Query sequence:', arg)
91         elif opt in ('-s', '-s'):
92             subject = arg
93             print('Subjuct sequence:',arg)
94         elif opt in ('--help', '-h'):
95             usage() # it doesn't work in getopt.getopt
96             sys.exit() # default is zero, means "successful termination", 2 means command line
97             errors abnormal termination, 1 means other abnormal termination
98     DP(query,subject)

```

Listing 2: Python Code