Yu Fu individual project summary. O56922. May 11, 2017.

Affine gap alignment

In homework 7, we did the global alignment of two strings with a fixed indel penalty. However, when there is a gap in either string that was extended over two nucleotides, the biological interpretation is still an indel, but only longer. Therefore, is it more realistic to penalize gap extension less than that of opening a gap.

My code is based on the global alignment algorithm from homework 7. So, I simply copied the global alignment function and then stated to code for an affine gap alignment. In my python file, I have two sets of strings. I used a pair of toy strings to test whether my function works or not and used the actual sequence of PDGF and Vsis gene from UCSC genome browser (without any trimming, because they are not that long) to run the alignments.

```
string1 = 'AATTCTTTCGC'
string2 = 'AACCC'
PDGF = 'GGGGGTGCAGGAAGCTCAACCCCTCTGGG...
Vsis = 'CTCGCTGCAAAGAGAAAACCGGAGCAGCC...
```

Code structure:

My affine gap alignment function consists of 3 parts:

1. Initialize the tables, including three scoring table (upper, mid, lower) and a backtrack table

```
midtable = initializeTable(len(string1)+1,len(string2)+1)
midtable[0][0]=0
midtable[0][1]= -openGapPenalty
midtable[1][0]= -openGapPenalty
for i in range(2,len(midtable)):
    midtable[i][0] = midtable[i-1][0] - gapExtensionPenalty
for j in range(2,len(midtable[0])):
    midtable[0][j] = midtable[0][j-1] - gapExtensionPenalty

##Upper table = edges indicates gap extension in string 1
uppertable = initializeTable(len(string1)+1,len(string2)+1)
for i in range(len(midtable)):
    uppertable[i][0] = len(string1)*-openGapPenalty*10000000
    uppertable[0][1] = -openGapPenalty
for i in range(2, len(uppertable[0])):
    uppertable[0][i] = uppertable[0][i-1] - gapExtensionPenalty

##Lower table = edges indicates gap extension in string 2
lowertable = initializeTable(len(string1)+1,len(string2)+1)
for i in range(len(midtable[0])):
    lowertable[0][i] = len(string1)*-openGapPenalty*1000000
lowertable[1][0] = -openGapPenalty
for i in range(2, len(midtable)):
    lowertable[i][0] = lowertable[i-1][0] - gapExtensionPenalty

## Initialize backtrack table
backtracktable = initializeTable(len(string1)+1,len(string2)+1)
for i in range(len(backtracktable[0])):
    backtracktable[0][i] = 'e'
for i in range(len(backtracktable)):
    backtracktable[i][0] = 's'
backtracktable[0][0] = '*'
```

2. Fill the scoring tables and backtrack table

```
#filling the affine gap alignment tables
for i in range(1,len(midtable)):
    for j in range(1,len(midtable[i])):
        lowertable[i][j] = max(lowertable[i-1][j] - gapExtensionPenalty, midtable[i-1][j] - openGapPenalty)
        uppertable[i][j] = max(uppertable[i][j-1] - gapExtensionPenalty, midtable[i][j-1] - openGapPenalty)
        score1 = midtable[i-1][j-1]
        # match
        if string1[i-1] == string2[j-1]:
            w1 = matchScore
        # mismatch
        if string1[i-1] != string2[j-1]:
            w1 = -mismatchPenalty
        # make the best choice
        midtable[i][j] = max(score1+w1,lowertable[i][j],uppertable[i][j])

#filling backtrack table
        decision = max(score1+w1,lowertable[i][j],uppertable[i][j])
        if decision == score1 + w1:
            backtracktable[i][j] = 'd'
        elif decision == lowertable[i][j]:
            backtracktable[i][j] = 's'
        elif decision == uppertable[i][j]:
            backtracktable[i][j] = 'e'
```

3. Re-generate the alignment from the backtrack table

```
## table to alighment code from homework 7
a1 = ''
a2 = ''
i = len(string1)
j = len(string2)
while i > 0 or j > 0:
    if backtracktable[i][j] == 'd':
        a1 = string1[i-1] + a1
        a2 = string2[j-1] + a2
        i = i-1
        j = j-1
    elif backtracktable[i][j] == 's':
        a1 = string1[i-1] + a1
        a2 = '-' + a2
        i = i-1
    elif backtracktable[i][j] == 'e':
        a1 = '-' + a1
        a2 = string2[j-1] + a2
        j = j-1
```

Results:

This is the scoring rules:

```
matchScore = 1
mismatchPenalty = 1
indelScore = 10
openGapPenalty = 10
gapExtensionPenalty = 1
```

The result of the test strings:

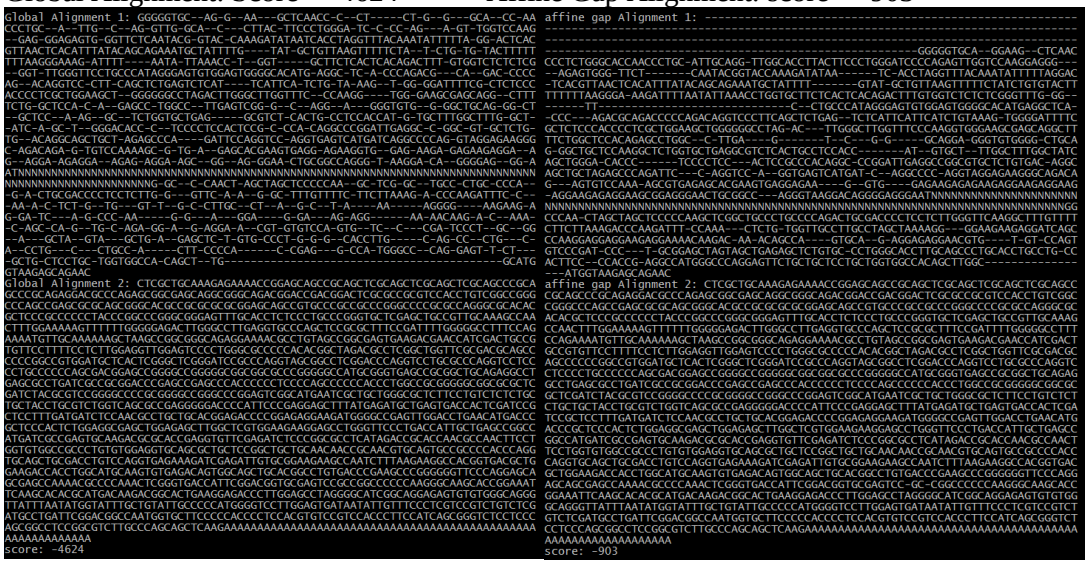
```
$ py AffineGapAlignment.py
string1: AATTCTTTCGC
string2: AACCC

Global Alignment 1: AATTCTTTCGC
Global Alignment 2: AA--C---C-C
score: -55

affine gap Alignment 1: AATTCTTTCGC
affine gap Alignment 2: AA------CCC
score: -12
```

- Affine gap alignment has a score of -12, only one gap
- Global alignment has a score of -55, three gaps
- Affine gap alignment is an algorithm that favors long single gaps instead of short single ones.
- Thus, it works.

Then, I moved on to align PDGF and Vsis
The results:
Global Alignment: Score = -4624 Affine Gap Alignment: score = -903



Interpretation:

- The global alignment obviously has more random gaps then the affine gap alignment. From the affine gap alignment, we can observe a few chunks of match/mismatch nucleotides without any gaps within, and there is a huge gap at the beginning of PDGF. They both show that the PDGF is to certain extent conservative (but with SNPs) to Vsis, or the other way around depending on which gene occurs first evolutionarily.

To Improve:

- Simplify the algorithm: It took my computer 3 minutes to run the alignments for PDGF and Vsis. It seems ok but if the sequence is longer, then it would probably take much more time to run.
- Try different sequences. I have previous knowledge that there is relatedness between PDGF and Vsis, and the results is predicted. What if there are two random sequences? How to determine that the alignments are biologically reasonable? In other words, how long the chunk in affine gap alignment should be in order to have an actual biological interpretation?
- Is there a better scoring matrix?