Ruby Blaser

Rblaser1

**Pairwise Sequence Alignment Tool**

Background:

Pairwise sequence alignment is an essential tool in bioinformatics to compare two DNA or protein sequences in order to try to determine homology between the two sequences. Finding the closest, most optimal alignment is always the goal in pairwise sequences alignment. There are also two different kinds of pairwise alignment: local and global. Global alignment analyzes and aligns the entire sequence, matching as many amino acids or nucleotides as possible. Local alignment is a more simplified version that finds areas in the sequences where there a high level of matches, and aligns those, creating “islands” of alignment matches. In order to be more thorough, I will be implementing global alignment in my sequence alignment tool. For global pairwise sequence alignment, my alignment code will be implementing the Needleman-Wunsch algorithm in Python. In this algorithm, a 2D matrix is used to store alignment scores, and a separate scoring system is used to score the alignment. In my tool, I will be using the BLOSUM62 matrix. This matrix is based on conserved amino acid patterns in the two sequences. A direct observation is made of the conserved regions, and a log(observed/expected) formula is used to generate the matrix. This particular matrix is actually best used for two sequences that are not that closely related as it only works for sequences that are less than 62% identical (hence BLOSUM62). Gap penalties are also assigned in order to complete the alignment score. Gap penalties are score penalties taken away from the overall score when gaps are found in the alignment. There are opening gap penalties for when the first gap is introduced, and subsequent existence and extension penalties for further gaps.

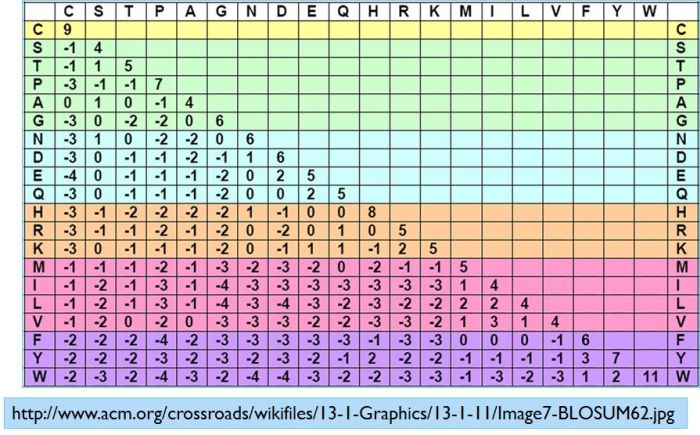
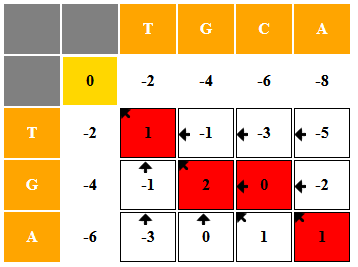


Figure 2 Needleman-Wunsch Algorithm

Figure 1 BLOSUM62 Matrix

Functionality: In this tool, users will be asked to input two DNA sequences, two protein sequence, or two NCBI Entrez accession numbers for alignment. The simplest scenario will be two protein sequences, which will simply be run through the Needleman-Wunsch alignment algorithm, and the most optimal alignment of the two sequences will be printed to output. If the user enters DNA sequences, an additional function will be used to convert these sequences to protein sequences before performing the alignment, since the BLOSUM62 matrix only scores amino acid sequences. If they enter the NCBI accession numbers, the program will query Entrez for the protein sequences, and proceed accordingly, returning the optimal alignment.

Development: First, for the initial input, I will use HTML forms coding to create a website with a large box for input of the sequences or accession numbers. There will also be buttons where the user can indicate if they are DNA or protein sequences. Using a CGI script, this input will then be sent through a Python program for alignment as described above, and return the alignment output to the user on the .cgi web page. For this alignment, I will also assign the gap penalties so they are hard coded. I will stick to the traditional BLOSUM62 gap penalties of 11/1, meaning an opening gap penalty of 11 and an extension penalty of 1. There are built in tools in the Bio package in Python such as Bio.align that could do this work for me, but I have enjoyed learning more about the Needleman-Wunsch algorithm, and how to implement that myself to do the work. I also plan to use CSS scripting to format the box and buttons, and make the returned alignment look nice, and be color coded. All in all, this may seem like a fairly simple concept, but it will take a lot of different working parts to actually implement.

Obviously, I will need to modify this code to incorporate the BLOSUM62 matrix, but here is a basic implementation of the Needleman-Wunsch algorithm in the Python language:

def needleman\_wunsch(seq1, seq2, match\_score=1, mismatch\_score=-1, gap\_penalty=-1):

n = len(seq1)

m = len(seq2)

# Initialize the scoring matrix

score\_matrix = [[0 for \_ in range(m + 1)] for \_ in range(n + 1)]

for i in range(n + 1):

score\_matrix[i][0] = gap\_penalty \* i

for j in range(m + 1):

score\_matrix[0][j] = gap\_penalty \* j

# Fill the scoring matrix

for i in range(1, n + 1):

for j in range(1, m + 1):

match = score\_matrix[i - 1][j - 1] + (match\_score if seq1[i - 1] == seq2[j - 1] else mismatch\_score)

delete = score\_matrix[i - 1][j] + gap\_penalty

insert = score\_matrix[i][j - 1] + gap\_penalty

score\_matrix[i][j] = max(match, delete, insert)

# Traceback to reconstruct the alignment

aligned\_seq1 = ""

aligned\_seq2 = ""

i, j = n, m

while i > 0 or j > 0:

if i > 0 and j > 0 and score\_matrix[i][j] == score\_matrix[i - 1][j - 1] + (match\_score if seq1[i - 1] == seq2[j - 1] else mismatch\_score):

aligned\_seq1 = seq1[i - 1] + aligned\_seq1

aligned\_seq2 = seq2[j - 1] + aligned\_seq2

i -= 1

j -= 1

elif i > 0 and score\_matrix[i][j] == score\_matrix[i - 1][j] + gap\_penalty:

aligned\_seq1 = seq1[i - 1] + aligned\_seq1

aligned\_seq2 = "-" + aligned\_seq2

i -= 1

else:

aligned\_seq1 = "-" + aligned\_seq1

aligned\_seq2 = seq2[j - 1] + aligned\_seq2

j -= 1

return aligned\_seq1, aligned\_seq2, score\_matrix[n][m]

