BA ICA1 217754 Affine Gaps

Section 1: A discussion on AGP

Numerous incarnations of sequence-based biology and ranging definitions of similarity have led to the derivation of purpose-tailored methods to align subject sequences, primarily geared towards inferring similarity (and ultimately shared downstream biological function), finding evolutionary relationships, genome assembly and repeat identification (Sung, 2009). Local alignment, used to find common substrings in two sequences, has several variations that can differ by the scoring system applied to maintain adequate algorithm performance. The brute-force approach (using no scoring system), involves finding all possible shared substrings in sequences, deriving the global alignment between each pair of substrings by lining them manually and manually finding the highest-scoring global alignment. However, it should be clear that any scoring system would be preferable to this, as the performance of such an algorithm is $O\binom{n}{2}\binom{m}{2}nm = On^3m^3$ (very poor) where n and m are the lengths of sequence x and y. Scoring systems include 'gap penalties', which determine the score reduction for the presence of a gap in either sequence.

Linear gap penalties (which employ a penalty for each gap in the alignment $penalty = gl$ where g is gap penalty and l is the length of the gap) can be used for a superficial comparison of DNA or Protein sequences in $O(mn)$ time. However, the linearity of this system (named because the penalty is fixed and does not adjust/reduce with gap extension means that information on key biological mechanisms may not be captured. A general gap penalty g could be used to penalise the score of an alignment with gaps, however, this still does not reflect some of the intricacies of mutational events. Specifically, there is a tendency for insertions or gaps in sequences to arise in DNA from mutational events in length k rather than k single gaps. Similarly, when aligning mRNA strings to the pre-transcript sequences, it is more likely to observe gaps defined k length rather than k gaps, as the former is likely to describe the spliced intronic regions which do not appear in the mRNA stretch (Sung, 2009).

The penalty scheme which addresses this tendency is known as the 'affine gap' penalty system.

The affine gap penalty scheme (AGP) has two (typically negative) penalty values: one for creating gap h and one for extending gaps g, and relies on the penalty for creating a gap being more severe (more negative value) than that for extending gaps. The overall penalty for a gap is therefore $penalty = h + gs$, where s is the length of the gap. To preserve a performance of $O(mn)$ in dynamic programming, the use of additional penalties requires a calculation of three matrices rather than one: ***M***, recording matching paths or insertions in either sequence, ***Ix*** which records the opening and extension of gaps in 'sequence x', and ***Iy***, which records gaps in 'sequence y'. AGP can be tailored further to suit the sequences under scrutiny by adjusting penalty numbers, for example decreasing h to allow more distantly related sequences to align (Madhusudhan *et al.,* 2006).

$$M(i,j) = \max\begin{cases} M(i-1,j-1) + s(x_i, y_j) & \text{match } x_i \text{ with } y_j \\ I_x(i-1,j-1) + s(x_i, y_j) & \text{insertion in } x \\ I_y(i-1,j-1) + s(x_i, y_j) & \text{insertion in } y \end{cases}$$

$$I_x(i,j) = \max\begin{cases} M(i-1,j) + h + g & \text{open gap in } x \\ I_x(i-1,j) + g & \text{extend gap in } x \end{cases}$$

$$I_y(i,j) = \max\begin{cases} M(i,j-1) + h + g & \text{open gap in } y \\ I_y(i,j-1) + g & \text{extend gap in } y \end{cases}$$

*Figure 1-The recursion tool for calculating the three matrix values in affine gap penalty scheme. **I** denotes position in seq x, **j** denotes position in seq y. (Settles, 2008)*

Efforts to represent the true biology of indel prevalence in sequence alignment have led to alternate gap penalty scoring systems to AGP including probabilistic, variable, convex and log scoring methods.

A convex penalty scheme decreases the penalty of a subsequent gap as a gap is extended, meaning that long gaps are not penalised based on their length (reflecting biological rule) (Waterman, 1984). Later literature showed that the efforts to achieve biological representation had led to less accurate alignments than affine gaps and paired with the $O(mn(n+m))$ performance, the use of convex penalties is not recommended.

As indel size appears to be distributed under a power law, methods were conceived to use ***log penalties*** (another convex scheme) (Benner, Cohen and Gonnet, 1993). The community disregarded the slower performance of log penalties for a higher accuracy than affine penalties, however, this benefit

was disproven by Cartwright, who concluded “the power law does not imply that gap costs should be logarithmic, instead, it implies that gap costs should be log-affine” (Cartwright, 2006)

Alternatively, AGP may not be used in a specific case such as the comparison of spliced and non-spliced DNA as shown in the note by *Mott,* which introduces terms for intronic regions (Mott, 1997)

AGP are often replaced by alternative methods during protein sequence alignment due to residue behaviour. In the simplest case, modifications to the affine gap model consider protein domain conservation patterns by assuming that conserved residues appear largely uninterrupted but are flanked by areas of non-conserved regions (Altschul, 1998). However, more explicit approaches show that protein sequence structure can even be taken into account, as the variable gap penalty system described by (Madhusudhan *et al.,* 2006), which showed that indels in regions of characterised secondary structure (buried/3D distance in folded protein) can be penalised more heavily than those in non-annotated stretches.

Block substitution Matrix (BLOSUM) and Point Accepted Mutation (PAM) score the likelihood of substitutions based on residue physicochemical properties, reflecting the different rates of substitution to each amino acid. This does not directly replace AGP but alters the alignments presented significantly as the comparison tool is far more complex than match/mismatch.

Section 2: Adapted class code

Please consult the code file for the full picture of what is going on, snippets here give highlights of functionality.

Added `argparse` arguments for the user to input the desired gap extension and opening penalties, as well as the scores for matching or non-matching bases. These scoring numbers were assigned to variables and a comparison was made to ensure that the penalty for opening a gap was larger than that for extending a penalty (if not the programme ends).

```
parser = argparse.ArgumentParser(description='Aligning sequences...')
parser.add_argument('seq1',action="store",help="First sequence")
parser.add_argument('seq2',action="store",help="Second sequence")
parser.add_argument('extend_gap',action="store",help="Penalty for gap extension",type =int) #NOTE take gap extension penalties fro
parser.add_argument('open_gap',action="store",help="Penalty for gap creation",type =int) #NOTE take gap creation penalties from th
parser.add_argument('seqmatch',action="store",help="Score for matching bases, +1 recommended for DNA",type =int,default=+1) #NOTE
parser.add_argument('seqmismatch',action="store",help="Score for non-matching bases, -1 recommended for DNA",type =int,default=-1)
```

`Initialise_matrices()` creates a list of lists as before, but each ‘cell’ holds an additional list of three values (representing three matrices **M, Ix** and **Iy** used to track gap opening and extension). These are updated from 0 to `[0,-inf,-inf]` in the first row and column of the matrix, which is aligned to a padding gap at the start of each sequence.

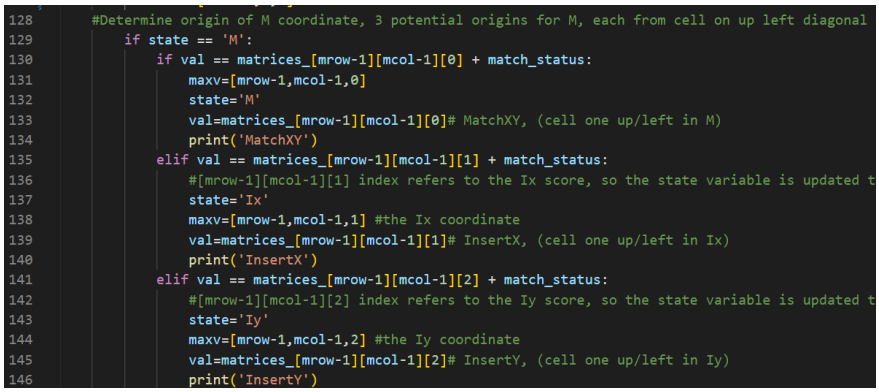
```
#make the matrices' overall structure NOTE this is a major change from standard SW
def initialise_matrices():
    global rows, cols
    mats=[0,0,0] #first position refers to matrix M, second Ix, third Iy
    matrices_ = [[mats for col in range(cols+1)] for row in range(rows+1)]
    # set Ix and Iy first column as negative infinity
    for row in range(0,rows+1):
        matrices_[row][0]=[0,float('-inf'),float('-inf')]
    # set Ix and Iy first row as negative infinity
    for col in range(0,cols+1):
        matrices_[0][col]=[0,float('-inf'),float('-inf')]
    return matrices_
```

`calc_score()` differs here because the recursion tool is different, totalling 7 comparisons including the addition of extend and open penalties. The calculations are labelled here with the alignment action they represent (`MatchXY`, `OpenGapX` etc.). This function is called by `build_matrix()` as in the class code version to build the initial scoring matrix used for traceback.

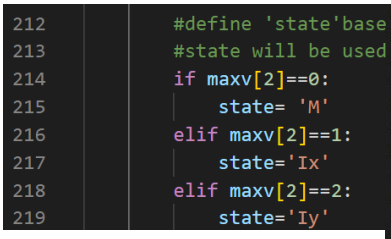
```
    MatchXY=(matrices_[row-1][col-1][0]+match_status) #(cell one up/left in M)
    InsertX=(matrices_[row-1][col-1][1]+match_status) #(cell one up/left in Ix)
    InsertY=(matrices_[row-1][col-1][2]+match_status) #(cell one up/left in Iy)
    M_val=max(0,MatchXY,InsertX,InsertY)

    OpenGapX=(matrices_[row-1][col][0]+g+h) #(cell one left in M)+gap_ext+gap_opn
    ExtendGapX=(matrices_[row-1][col][1]+g) #(cell one left in Ix)+gap_ext
    Ix_val=max(0,OpenGapX,ExtendGapX)

    OpenGapY=(matrices_[row][col-1][0]+g+h) #(cell one up in M)+gap_ext+gap_opn
    ExtendGapY=(matrices_[row][col-1][2]+g) #(cell one up in Iy)+gap_ext
    Iy_val=max(0,OpenGapY,ExtendGapY)
    #update the matrices'
    vals=[M_val,Ix_val,Iy_val]
    print(f"vals = {vals}")
    return vals
```

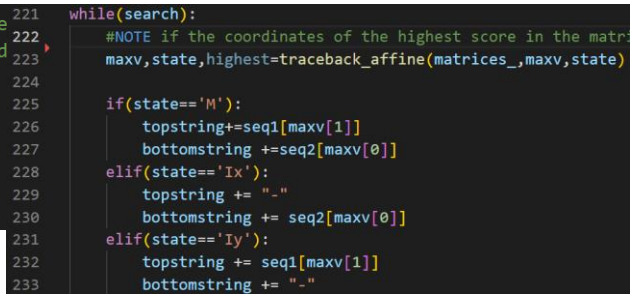
`get_max()` additionally loops through each cell's list of three values to find the highest value across the three matrices. The search was reversed to start from the bottom right cell in case there were several instances of the same score in the table, allowing more of the sequence to be captured in `print_traceback()` later (which starts at the end).

`traceback_affine()` is adjusted substantially, now taking the highest value of interest, checking what the score is (`val`), and comparing it to the potential origin cells by reiterating the recursion tool calculations. Gap penalties are only used here to trace back the path through the previously calculated scores. `state` keeps track of movement into each matrix, and helps to decide whether to extend a gap or not. Before line 128 the function creates a shortcut to input the rest of the opposing sequence when the edge of the traceback matrix has been reached. The returned `maxv` is the 3-number coordinates of the next value in the traceback path (see snippet for the derivation of **M** matrix origin).

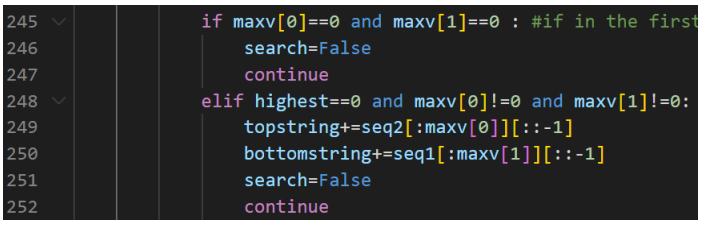


In `print_traceback()`, `maxv`'s final number is used to determine the current highest score's matrix, which determines the notation of the alignment instead of comparing it to the previous column and row indexes. Sequence letters are printed as shown opposite dependent on the matrix state.





The search will continue until it has regressed to the top left cell or a value of 0 is reached in the middle of the matric, setting the search variable to `False`. The rest of the sequence is printed too.



Despite all efforts, sequences preferred to produce a large gap at the start of the sequence rather than shift the bases to the end of the alignment and leave a gap in the middle. This undermines the purpose of using SW AGP, however, I struggled to determine the origin of this issue. My best suggestion is that the variable referencing in the recursion tool `calc_score()` and `traceback_affine()` is inconsistent, meaning that the incorrect path is chosen when a gap should be created (which would mean switching to **Ix** or **Iy** mid-path. Specifically, I think the issue is the use of a single `match_status` and not two match and mismatch variables in `calc_score()`, though I cannot determine the correction.

References

Altschul, S.F. (1998) 'Generalized affine gap costs for protein sequence alignment', *Proteins, structure, function, and bioinformatics,* 32(1), pp. 88-96. doi: 10.1002/(SICI)1097-0134(19980701)32:13.0.CO;2-J.

Benner, S.A., Cohen, M.A. and Gonnet, G.H. (1993) 'Empirical and Structural Models for Insertions and Deletions in the Divergent Evolution of Proteins', *Journal of molecular biology,* 229(4), pp. 1065-1082. doi: 10.1006/jmbi.1993.1105.

Cartwright, R.A. (2006) 'Logarithmic gap costs decrease alignment accuracy', *BMC Bioinformatics,* 7(1), pp. 527. doi: 10.1186/1471-2105-7-527.

Madhusudhan, M.S., Marti-Renom, M.A., Sanchez, R. and Sali, A. (2006) 'Variable gap penalty for protein sequence–structure alignment', *Protein engineering, design and selection,* 19(3), pp. 129-133. doi: 10.1093/protein/gzj005.

Mott, R. (1997) 'EST_GENOME: a program to align spliced DNA sequences to unspliced genomic DNA', *Bioinformatics,* 13(4), pp. 477-478. doi: 10.1093/bioinformatics/13.4.477.

Settles, B. (2008) *Sequence Alignment.* Available at: https://pages.cs.wisc.edu/~bsettles/ibs08/ (Accessed: Mar 16, 2023).

Sung, W. (2009) *Algorithms in Bioinformatics : A Practical Introduction.* Philadelphia, PA: CRC Press LLC.

Waterman, M.S. (1984) 'Efficient sequence alignment algorithms', *Journal of theoretical biology,* 108(3), pp. 333-337. doi: 10.1016/S0022-5193(84)80037-5.