

Review:

- Alignment for bio-sequence comparison

- Dynamic programming algorithm

- define subproblems: $D[i, j] = \text{optimal align score of } S[1..i], T[1..j]$

- recurrence relation

- base case

- order of computation

- backtracking

- practical tricks

- linear space score

- compute the "pointers" during backtracking

- gap penalty

- arbitrary $g(k)$

$$\begin{array}{cc} S[1..i-1] & S[i] & S[1..i-1]S[i] \\ T[1..j-1] & T[j] & T[1..j] \\ \hline D[i-1, j-1] + f(S[i], T[j]) & & D[i-1, j] + \text{indel} \end{array}$$

Affine gap penalty example

For example: match = 1; mismatch = -1; gap open = -5; gap extension = -1.

- ATAGG--AAG

- | | | | |

- ATTGGCAATG

- 6 match, 2 mismatch, 1 gap open, 2 gap extension, score = ?

- ATAGG-AA-G

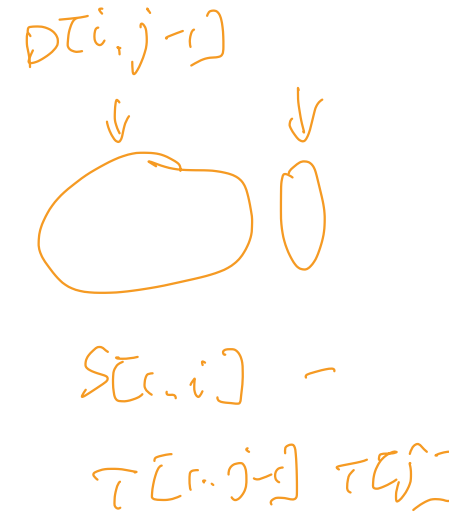
- | | | | |

- ATTGGCAATG

$$6 - 2 - 5 - 2 = -3$$

$$7 - 1 - 5 - 1 - 5 - 1 = -6$$

Old Algorithm Does not Work



- Consider the last column of an alignment again:



- When the last column is an indel, the added cost depends on the previous column.
 - If previous column has a gap opened already, then
 - $D[4,6] = D[4,5] + \text{gapext}$
 - Else
 - $D[4,6] = D[4,5] + \text{gapopen} + \text{gapext}$
- How do we know the previous column's configuration?
- Because by induction we know the optimal solution for $D[i,j-1]$, can we simple look at it and use the configuration?

Algorithm for Affine Gap

- We compute the optimal solution by limiting the last column to one of the following three configurations:

$S[i..v]$

$T[i..j]$

ATAGG

ATAGG-

ATAGGC

|| ||

|| ||

|| ||

ATTGG

ATTGGC

ATTGG-

$D_0[i,j]$

$D_1[i,j]$

$D_2[i,j]$

(last column must be

$S[i] \text{ v.s. } T[j]$)

- v.s. $T[j]$)

$S[i]$ v.s. -

- We only distinguish them by the last column, there is no constraint for columns before the last column.

ATAGG

ATAGG-

ATAGGC

|| |

|| |

|| |

ATTGG

ATTGGC

ATTGG-

$D_0[i,j]$

$D_1[i,j]$

$D_2[i,j]$

$S[1..i-1] S[i]$

$T[1..j-1] T[j]$

case 1

$S[1..i-2] S[i-1] S[i]$

$\rightarrow D_0[i,j] = D_0[i-1, j-1] + f(S[i], T[j])$

case 2

$T[1..j-2] T[j-1] T[j]$

$S[1..i-1] - S[i]$
 $T[1..j-2] T[j-1] T[j]$

$\Rightarrow D_0[i,j] = D_1[i-1, j-1] + f(S[i], T[j])$

case 3

$S[1..i-2] S[i-1] S[i]$
 $T[1..j-1] - T[j]$

$\rightarrow D_0[i,j] = D_2[i-1, j-1] + f(S[i], T[j])$

ATAGG

ATAGG-

ATAGGC

|| |

|| |

|| |

ATTGG

ATTGGC

ATTGG-

$D_0[i,j]$

$D_1[i,j]$

$D_2[i,j]$



$s[1..i] -$
 $t[1..j-1] t[j]$

$\rightarrow D_0[i,j-1] + \text{gapopen} + \text{gapext}$

case 1
 \rightarrow

$s[1..i-1] s[i]$
 $t[1..j-2] t[j-1] t[j]$

case 2
 \rightarrow

$s[1..i] -$
 $t[1..j-2] t[j-1] t[j]$

$\rightarrow D_1[i,j-1] + \text{gapext}$

case 3
 \rightarrow

$s[1..i-1] s[i]$
 $t[1..j-1] - t[j]$

$\rightarrow D_2[i,j-1] + \text{gapopen} + \text{gapext}$

Recurrence Relation

- initialize
- for i
- for j

$$D_0[i,j] = f(s[i], t[j]) + \max \begin{cases} D_0[i-1, j-1]; \\ D_1[i-1, j-1]; \\ D_2[i-1, j-1]; \end{cases}$$

apply the recursive relations
- output $\max(D_0[m,n], D_1[m,n], D_2[m,n])$

$$D_1[i,j] = \text{gapext} + \max \begin{cases} D_0[i, j-1] + \text{gapopen}; \\ D_1[i, j-1]; \\ D_2[i, j-1] + \text{gapopen}; \end{cases}$$



$$D_2[i,j] = \text{gapext} + \max \begin{cases} D_0[i-1, j] + \text{gapopen}; \\ D_1[i-1, j] + \text{gapopen}; \\ D_2[i-1, j]; \end{cases}$$

Note the grayed cases can't be optimal so can be safely removed.

Algorithm

- No difference to the simple DP but now uses three arrays.
- Backtracking should be very careful!
- Still $O(nm)$ time. Approximately 3 times slower.
- This is okay because the model is more expressive.
- Much faster than the general gap penalty.

Gotoh, O., 1982. An improved algorithm for matching biological sequences. *Journal of molecular biology*, 162(3), pp.705-708.