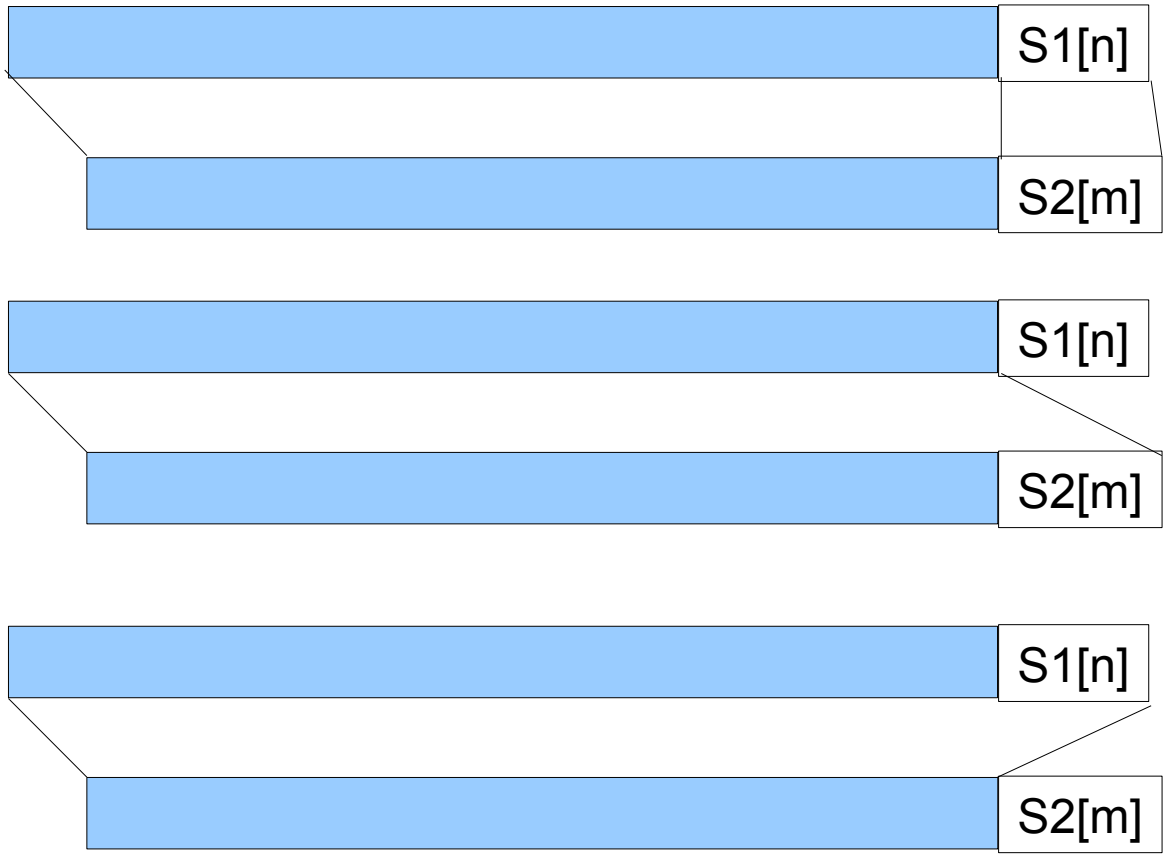
CMSC423: Bioinformatic Algorithms, Databases and Tools
Lecture 10

inexact alignment

dynamic programming, gapped alignment

Intuition

- What is the best way to align strings S1 and S2?
- just look at last character for now – what is it aligned to?



The recurrences

AG-C-GTAG
-GTCAG-A-

Score[i,j] is the maximum of:

1. Score[i-1, j-1] + Value[S1[i],S2[j]]

```
AG-C-G        AG-C-G
-GTCAG        -GTCAT
```

2. Score[i – 1, j] + Value[S1[i], -] (S1[i] aligned to gap)

```
AG-C-GT
-GTCAG-
```

3. Score[i, j – 1] + Value[-, S2[j]] (S2[j] aligned to gap)

```
AG-C-
-GTCA
```

The dynamic programming table

Score[i,j] is the maximum of:

1. Score[i-1, j-1] + Value(S1[i],S2[j]) (S1[i-1], S2[j-1] aligned)
2. Score[i – 1, j] + Value(S1[i], -) (S1[i] aligned to gap)
3. Score[i, j – 1] + Value(-, S2[j]) (S2[j] aligned to gap)

	-	A	G	C	G	T	A	G
-	0	-2	-4	-6	-8	-10	-12	-14
G	-2	-4	8	6				
T	-4	-6	6	4				
C	-6	-8	4	16				
A	-8							
G	-10							
A	-14							

Value (A, A) = 10
Value (A, G) = -5
Value (A, -) = -2

Note: we only look at 3 adjacent boxes

How do you output the result?

- Goal: produce the “nice” string with gaps that is shown in the examples
- Idea: create the string backwards – starting from the right
- As you follow backtrack pointers:
  - if you follow diagonal pointer – add characters to both output strings (aligned versions of original strings)
  - if you move up – add gap character to string represented on the y axis, add string character to string represented on x axis
  - if you move left – gap goes in string on x axis and character in string on y axis
- When you reach (0,0) output the two aligned strings

Local vs. global alignment

- Can we change the algorithm to allow S1 to be a substring of S2?

```
ACAGTTGACCCGTGCAT
----TG-CC-G------
```

- Key idea: gaps at the end of S2 are free
- Simply change the first row in the DP table to 0s
- Answer is no longer Score[n, m], rather the largest value in the last row

Sub-string alignment

	-	A	G	C	G	T	A	G
-	0	0	0	0	0	0	0	0
C	-2			10	8			
G	-4			8	20	18		
T	-6			6	18	30	28	26

AGCGTAG

CGT

Local alignment

- What if we just want a region of similarity?

```
ACAGTTGACCCGTGCAT
    ||  ||  |
GTCATG-CC-GAGATCG
```

- First row and column set to 0s
- Allow alignment to start anywhere:

Score[i,j] = max{0, case 1, case 2, case 3}

- Answer is location in matrix with highest score

Local alignment

		A	G	C	G	T	A	G
	0	0	0	0	0	0	0	0
C	0							
T	0		0					
C	0			10				
G	0				20			
T	0					30		
C	0							

```
AGCGTAG
  |||
CTCGTC
```

Various flavors of alignment

- Alignment problem also called "edit distance" – how many changes do you have to make to a string to convert it into another one.
- Edit distance also called Levenshtein distance
- Local alignment – Smith-Waterman
- Global alignment – Needleman-Wunsch

Gap penalties

How much do we pay for gaps?

- In the edit-distance/alignment framework

Cost(n gaps in a row) = n * Cost(gap)

- This doesn't work for e.g. RNA-DNA alignments

```
ACAGTTCGACTAGAGGACCTAGACCACTCTGT
    TTCGA----------TAGACCAC
```

- Affine gap penalties

Cost(n gaps in a row) = Cost(gap open) + n * Cost(gap)

- Gap opening penalty is high, gap extension penalty is low (once we start a gap we might as well pile more gaps on top)

Dynamic programming solution

- Traditional 1-table approach doesn't work anymore
- Instead, use 4 tables:
  - V – stores value of best alignment between S1[1..i], S2[1..j]
  - G – best alignment between S1[1..i], S2[1..j] s.t. S1[i] aligned with S2[j]
  - E – best alignment between S1[1..i], S2[1..j], s.t. alignment ends with gap in S1
  - F – best alignment between S1[1..i], S2[1..j], s.t. alignment ends with gap in S2
- V[i,j] = max(E[i,j], F[i,j], G[i,j])
- As in traditional approach, find box in V matrix where V[i,j] is maximal.

Affine gap recurrences

- V[i,j] = max[E[i,j], F[i,j], G[i,j]]
- G[i,j] = V[i-1, j-1] + Value(S1[i], S2[j])
  - irrespective how we got here (hence use of V), S1[i] and S2[j] are matched
- E[i,j] = max{E[i, j-1], V[i, j-1] – GapOpen} – GapExtend
  - either we add a gap in S1 to an existing one (E-GapExtend)
  - or we add a gap in S1 when there was none (V-GapOpen-GapExtend)
- F[i,j] = max{F[i-1, j], V[i-1, j] – GapOpen} – GapExtend
  - either we add a gap in S2 to an existing one (F–GapExtend)
  - or we add a gap in S2 when there was none (V-GapOpen-GapExtend)

Running times

- All these algorithms run in O(mn) – quadratic time
- Note – this is significantly worse than exact matching
- Next we'll talk about speed-up opportunities

- BTW, how much space is needed?

- If we only need to find the best score (not the exact alignment as well) – O(min(m,n))

- If we need to find the best alignment – elegant divide and conquer algorithm leads to linear space solution.