CMSC423: Bioinformatic Algorithms, Databases and Tools
Lecture 8

Sequence alignment: inexact alignment

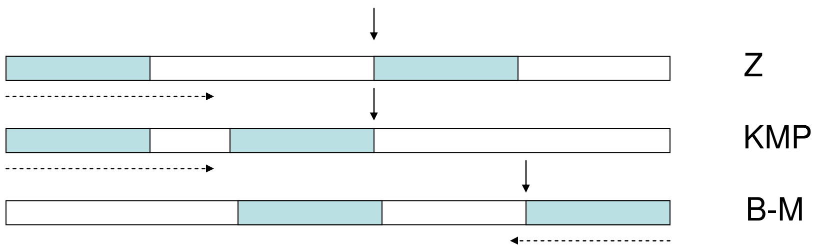
dynamic programming, gapped alignment

Note

- Lecture 7 – suffix trees and suffix arrays will be rescheduled

Exact alignment recap

- Exact matching can be done efficiently: O(|Text| + |Pattern|)
- Key idea: preprocess data to keep track of similar regions, then use information to "jump" over places where no match can occur

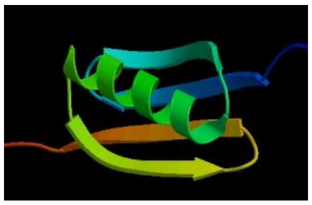


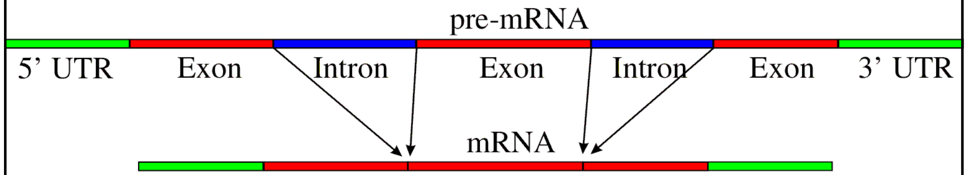
Inexact matching: why?

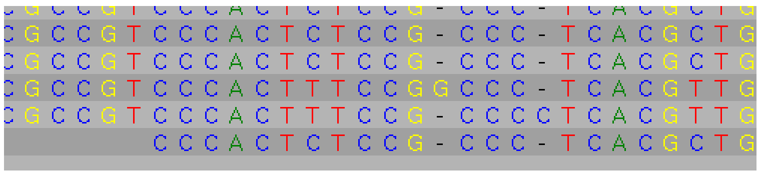
- Redundancy in genetic code: nucleotide sequence may differ, but proteins the same

```
S  Y  P  T  D
TCTTATCCTACTGAT
TCATACCCCACAGAC
```

- Different amino-acid sequences still fold the same way: function unchanged (generally changing an amino-acid with a similar one doesn't affect protein function)
- Aligning ESTs (RNA sequences) to DNA need to account for gaps corresponding to exons
- Need to account for sequencing errors







```
C G C C G T C C C A C T C T C C G - C C C - T C A C G C T G
C G C C G T C C C A C T C T C C G - C C C - T C A C G C T G
C G C C G T C C C A C T T T C C G G C C C - T C A C G T T G
C G C C G T C C C A C T T T C C G - C C C C T C A C G T T G
          C C C A C T C T C C G - C C C - T C A C G C T G
```

Several hemoglobins

```
HBB_HUMAN      FFESFGDLSTPDAVMGNPKVKAHGKKVL-----GAFSDGLAHLDNLKGTF
HBB_HORSE      FFDSFGDLSNPGAVMGNPKVKAHGKKVL-----HSFGEGVHHLDNLKGTF
HBA_HUMAN      YFPHF-DLS-----HGSAQVKGHGKKVA-----DALTNAVAHVDDMPNAL
HBA_HORSE      YFPHF-DLS-----HGSAQVKAHGKKVG-----DALTLAVGHLDDLPGAL
MYG_PHYCA      KFDRFKHLKTEAEMKASEDLKKHGVTVL-----TALGAILKKKGHHEAEL
GLB5_PETMA     FFPKFKGLTTADQLKKSADVRWHAERII-----NAVNDAVASMDDTEKMS
LGB2_LUPLU     LFSFLKGTSEVP--QNNPELQAHAGKVFKLVYEAAIQLQVTGVVVTDATL
                 *  :   .         .  .:: *.   :        :.    :
```

From http://bioinfo.cnio.es/docus/courses/SEK2003Filogenias/seq_analysis/multiple.html

Warm-up – Longest Common Subsequence

- Given two strings of letters, identify longest string of letters that occurs, in the same order, in both strings

```
AG C GTAG
 G C G A
 GTCAG A
```

	A	G	C	G	T	A	G
G		1		1			1
T					1		
C			1				
A	1					1	
G		1		1			1
A	1					1	

- Find the longest chain of 1s, moving to the right and down

Dynamic programming

- Idea: re-use previously computed information
- LCS[i,j] – longest common subsequence of strings S1[1..i], S2[1..j]

i

j

	A	G	C	G	T	A	G
G		1		1			1
T					1		
C			1				
A	1					1	
G		1		1			1
A	1					1	

Goal: find LCS[m,n]

LCS[i,j] is the maximum of:

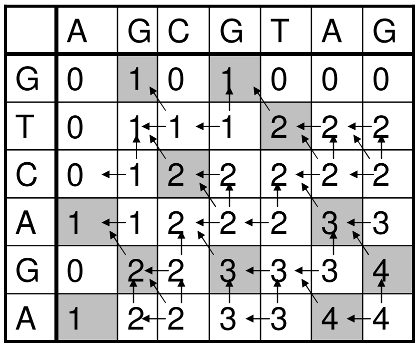
```
1.if S1[i] = S2[j]
    LCS[i-1, j-1] + 1
  else
    LCS[i -1, j-1]
2. LCS[i – 1, j]
3. LCS[i, j – 1]
```

Computing the LCS table

Row 0 and column 0 easy to fill
Fill the rest column by column

Find the actual sequence:
trace-back pointers

	A	G	C	G	T	A	G
G	0	1	0	1	0	0	0
T	0	1					
C	0	1					
A	1	1					
G	0	2					
A	1	2					



	A	G	C	G	T	A	G
G	0	1	0	1	0	0	0
T	0	1	1	1	2	2	2
C	0	1	2	2	2	2	2
A	1	1	2	2	2	3	3
G	0	2	2	3	3	3	4
A	1	2	2	3	3	4	4

Extending to sequence alignment

```
AG–C–GTAG
–GTCAG–A–
```

- **In LCS, mis-alignments were free**
- **What happens if we pay for our "mistakes"? (this also allows us to account for "similar" amino-acids)**
  - **Value[A, A] = 10**
  - **Value[A,G] = -5**
  - **Value[A,-] = -2**
  - **etc.**
- **The same dynamic programming algorithm works!**

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-1],S2[j-1]] (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

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