Chapter 5: Inexact alignment

Where we are...

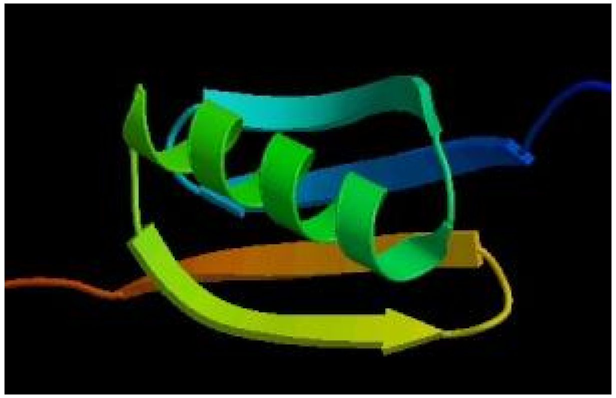
- We covered;
  - exact matching and k-mer counting (still exact)
  - probabilistic and branch and bound searching
- Now: inexact searching / alignment
  - longest common subsequence
  - alignment with arbitrary substitution matrices
  - gapped alignment with affine gap penalties
- Maybe:
  - reducing memory usage
  - RNA folding

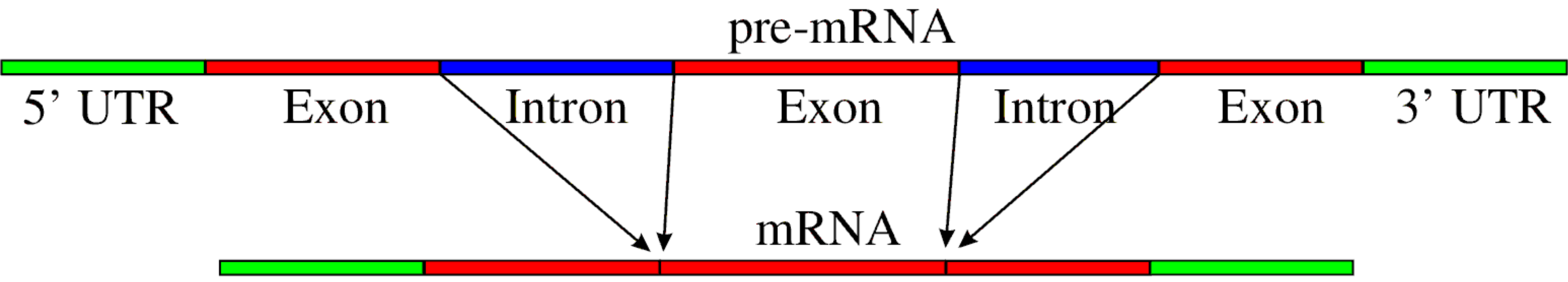
Inexact matching: why?

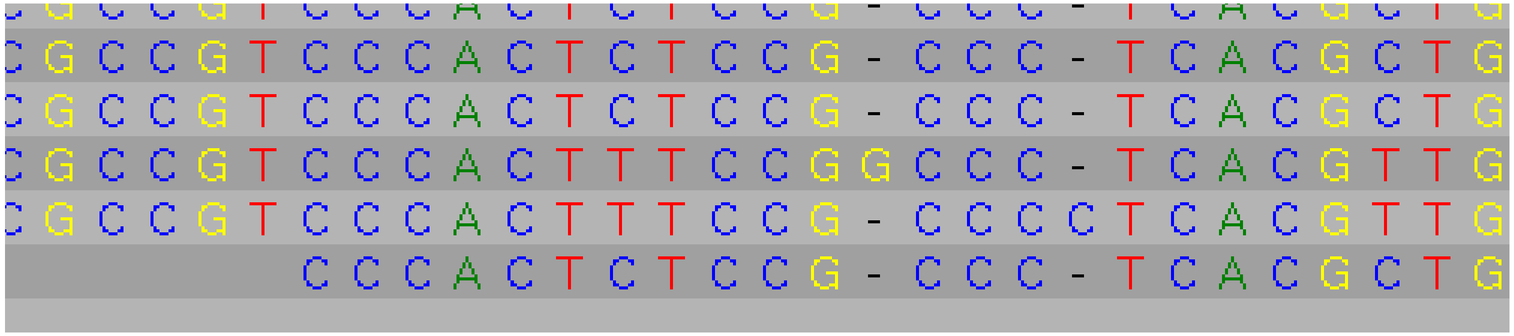
- Redundancy in genetic code: nucleotide sequence may differ, but proteins the same

```
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	

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]
```

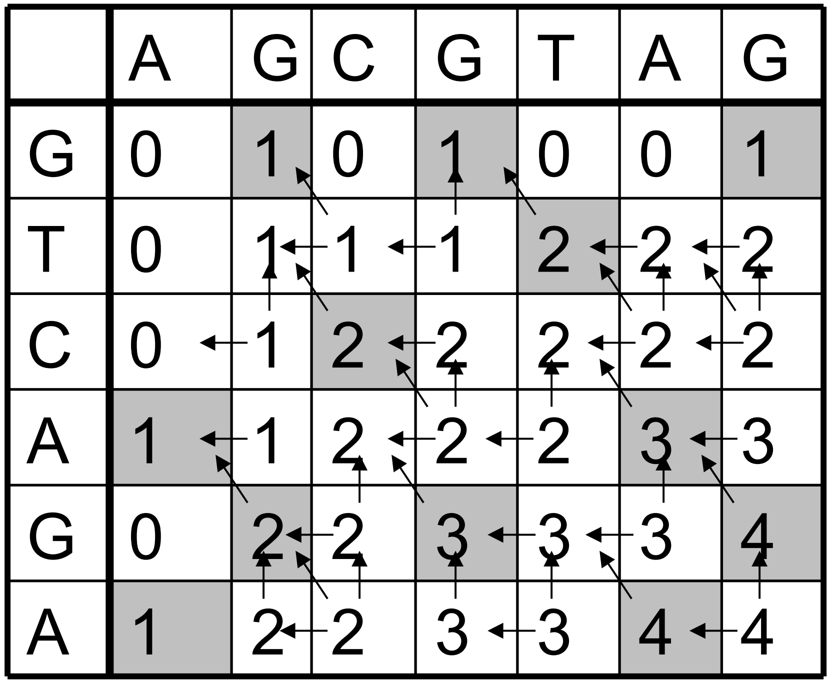
Goal: find LCS[m,n]

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	1
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	1
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],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	-12							

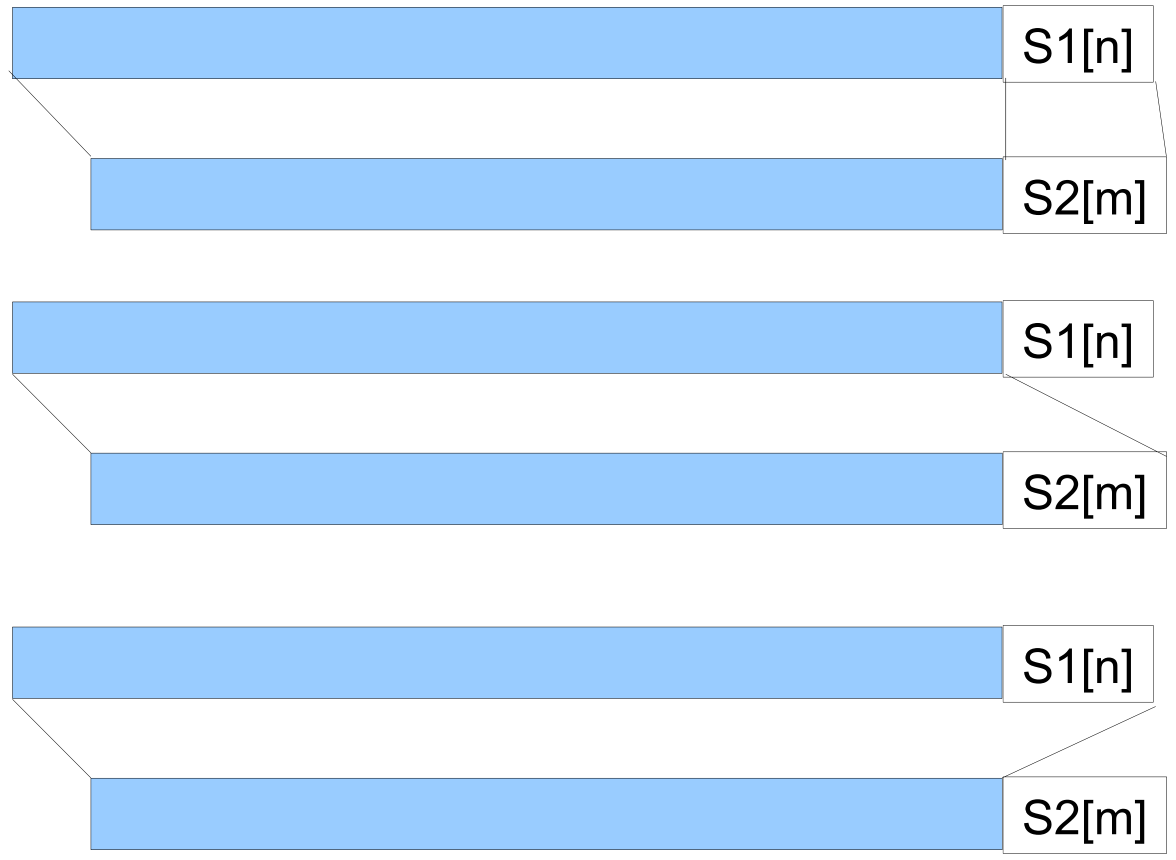
Value (A, A)	= 10
Value (A, G)	= -5
Value (A, -)	= -2

Note: we only look at 3 adjacent boxes

Intuition

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

```
AG-C-GTAG
-GTCAG-A-
```



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	-12							

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
```

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)

Arbitrary gap penalties

- Assume gap penalty given by function f(k) for gap of length k
- Can you modify dynamic programming to handle this situation?

Arbitrary gap penalties

- Assume gap penalty given by function f(k) for gap of length k
- Can change traditional dynamic programming as follows:

1. Score[i-1, j-1] + Value(S1[i],S2[j]) (S1[i-1], S2[j-1] aligned)
2. $\max_k$ Score[i – k, j] + f(k) (S1[i-k .. i] aligned to gaps)
3. $\max_k$ Score[i, j – k] + f(k) (S2[j-k .. j] aligned to gaps)

- Note: if f(a + b) > f(a) + f(b) we need to make sure Score[i-k, j] did not end in a gap – stay tuned

Edit distance with affine gaps

- Four dynamic programming tables
- V[i,j] – best alignment score of S_1[1..i], S_2[1..j]
- E[i,j] – best alignment score of S_1[1..i], S_2[1..j] when S_1[i] aligned to a gap
- F[i,j] – best alignment score of S_1[1..i], S_2[1..j] when S_2[j] aligned to a gap
- G[i,j] – best alignment score of S_1[1..i], S_2[1..j] when alignment ends without a gap

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

Recurrences

- $V[i,j] = \max(E[i,j], F[i,j], G[i,j])$
- $G[i,j] = V[i-1, j-1] + \text{score}(S_1[i], S_2[j])$
- $E[i,j]$ = maximum of
  - $V[i-1, j] + g_{open} + g_{extend}$
  - $E[i-1, j] + g_{extend}$
- $F[i,j]$ = maximum of
  - $V[i, j-1] + g_{open} + g_{extend}$
  - $F[i, j-1] + g_{extend}$

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.

Where do the alignment scores come from?

- PAM matrices
  - PAM1 – based on frequency of mutations between closely related proteins (within 1 "evolutionary step")
  - PAM 2 - ... within 2 evolutionary steps
  - ... PAM 250 – commonly used
- BLOSUM matrices
  - Frequency of mutations between proteins that are x% similar
  - BLOSUM100 – based on proteins that are exactly the same (e.g. score(A,A) is defined but not score(A,G))
  - BLOSUM62 – commonly used
- gap scores usually determined empirically

BLOSUM62

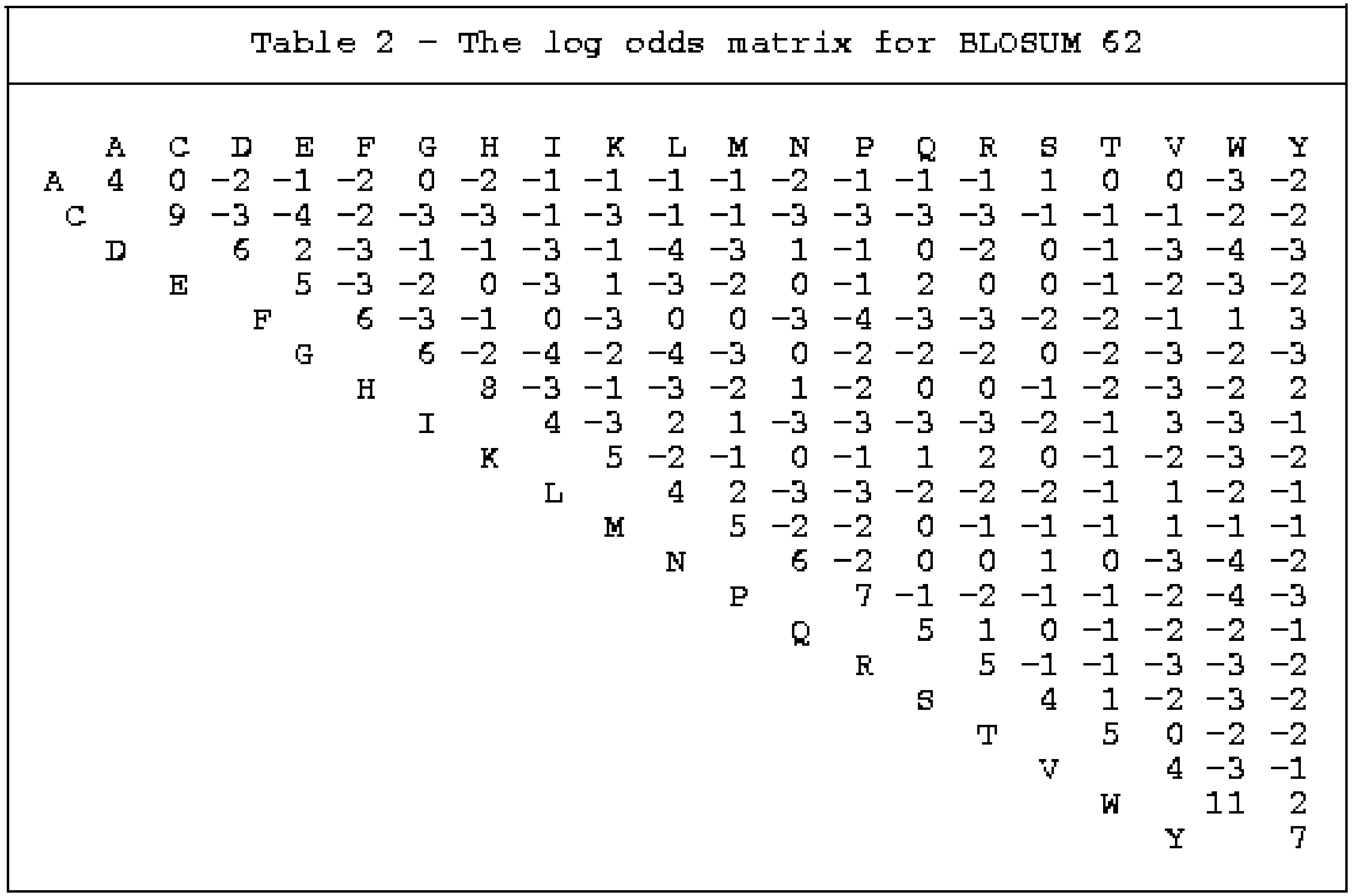


Table 2 – The log odds matrix for BLOSUM 62

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	4	0	−2	−1	−2	0	−2	−1	−1	−1	−1	−2	−1	−1	−1	1	0	0	−3	−2
C		9	−3	−4	−2	−3	−3	−1	−3	−1	−1	−3	−3	−3	−3	−1	−1	−1	−2	−2
D			6	2	−3	−1	−1	−3	−1	−4	−3	1	−1	0	−2	0	−1	−3	−4	−3
E				5	−3	−2	0	−3	1	−3	−2	0	−1	2	0	0	−1	−2	−3	−2
F					6	−3	−1	0	−3	0	0	−3	−4	−3	−3	−2	−2	−1	1	3
G						6	−2	−4	−2	−4	−3	0	−2	−2	−2	0	−2	−3	−2	−3
H							8	−3	−1	−3	−2	1	−2	0	0	−1	−2	−3	−2	2
I								4	−3	2	1	−3	−3	−3	−3	−2	−1	3	−3	−1
K									5	−2	−1	0	−1	1	2	0	−1	−2	−3	−2
L										4	2	−3	−3	−2	−2	−2	−1	1	−2	−1
M											5	−2	−2	0	−1	−1	−1	1	−1	−1
N												6	−2	0	0	1	0	−3	−4	−2
P													7	−1	−2	−1	−1	−2	−4	−3
Q														5	1	0	−1	−2	−2	−1
R															5	−1	−1	−3	−3	−2
S																4	1	−2	−3	−2
T																	5	0	−2	−2
V																		4	−3	−1
W																			11	2
Y																				7

Questions

- How would you modify the dynamic programming algorithm described in the class in order to perform alignments anchored at the beginning of the sequences (i.e. not penalize gaps at the end of the sequences)?
- How would you modify the dynamic programming algorithm to find 'overlap' alignments – the alignment must include one end from each of the sequence (see below)?