INT102
Algorithmic Foundations And Problem Solving
Sequence Alignment by Dynamic Programming

Dr Pengfei Fan
Department of Intelligent Science

西交利物浦大学
Xi'an Jiaotong-Liverpool University

Learning Outcomes

- Longest Common Subsequence (LCS)
- Pairwise Sequence Alignment Problem
  1. Global Alignment
  2. Local Alignment

DNA Sequence Comparison
A topic in Bioinformatics

- Finding sequence similarities with genes of known function is a common approach to infer a newly sequenced gene's function

- Gene similarities between two genes with known and unknown function alert biologists to some possibilities

- Computing a similarity score between two genes tells how likely it is that they have similar functions

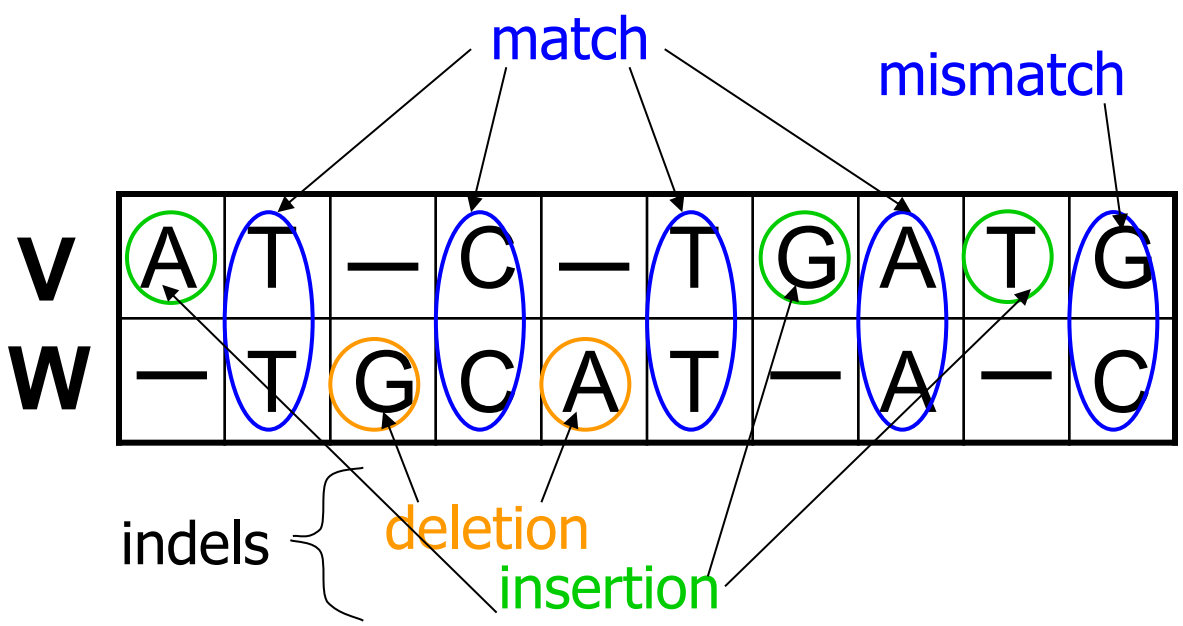
Aligning DNA Sequences

Alignment : 2 x ***k*** matrix (***k*** ≥ ***m***, ***n***)

v = ATCTGATG n = 8

w = TGCATAC m = 7

4 matches
1 mismatches
3 insertions
2 deletions



Longest Common Subsequence (LCS)

Longest Common Subsequence (LCS)

- DNA analysis, two DNA string comparison.
- DNA string: a sequence of symbols A,C,G,T.
  - S=ACCGGTCGAGCTTCGAAT
- Subsequence (of *X*): is *X* with some symbols left out.
  - *Z*=CGTC is a subsequence of *X*=ACGCTAC.
- Common subsequence *Z* (of *X* and *Y*): a subsequence of *X* and also a subsequence of *Y*.
  - *Z*=CGA is a common subsequence of both *X*=ACGCTAC and *Y*=CTGACA.
- Longest Common Subsequence (LCS): the longest one of common subsequences.
  - *Z'* =CGCA is the LCS of the above *X* and *Y*.
- LCS problem: given $X=\langle x_1, x_2,\ldots, x_m\rangle$ and $Y=\langle y_1, y_2,\ldots, y_n\rangle$, find their LCS.

LCS Intuitive Solution –brute force

- List all possible subsequences of X, check whether they are also subsequences of Y, keep the longer one each time.
- Each subsequence corresponds to a subset of the indices $\{1,2,\ldots,m\}$, there are 2^m. So exponential.
- Dynamic Programming (DP)

Dynamic Programming (DP)

- Initialization
- Recursive Solution
- Trace Back

Analysis

LCS DP – Step 1: Optimal Substructure

Let $X=<x_1, x_2,\ldots, x_m>$ $(= X_m)$ and

$Y=<y_1, y_2,\ldots,y_n>$ $(= Y_n)$

and

$Z=<z_1, z_2,\ldots, z_k>$ $(= Z_k)$ be any LCS of X and Y,

1. if $x_m= y_n$, then $z_k= x_m= y_n$, and Z_{k-1} is the LCS of X_{m-1} and Y_{n-1}.
2. if $x_m\neq y_n$, then $z_k\neq x_m$ implies Z is the LCS of X_{m-1} and Y_n.
3. if $x_m\neq y_n$, then $z_k\neq y_n$ implies Z is the LCS of X_m and Y_{n-1}.

For example:

X = ABCD	X = ABCF	X = ABCD
Y = ACD	Y = ABC	Y = ECDF
Z = ACD and D = D = D	Z = ABC and C ≠ F	Z = CD and D ≠ F

Analysis

LCS DP – Step 2: Recursive Solution

- What Step 1 says:
  - If $x_m = y_n$, find LCS of X_{m-1} and Y_{n-1}, then append x_m.
  - If $x_m \neq y_n$, find LCS of X_{m-1} and Y_n and LCS of X_m and Y_{n-1}, take which one is longer.
- Overlapping substructure:
  - Both LCS of X_{m-1} and Y_n and LCS of X_m and Y_{n-1} will need to solve LCS of X_{m-1} and Y_{n-1}.
- $c[i,j]$ is the length of LCS of X_i and Y_j .

$$c[i,j] = \begin{cases} 0 & \text{if } i=0, \text{ or } j=0 \\ c[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max\{c[i-1,j], c[i,j-1]\} & \text{if } i,j>0 \text{ and } x_i \neq y_j \end{cases}$$

LCS Intuitive Solution – Dynamic Programming

Longest Common Subsequence

X “aab”
Y “azb”

LCS(“aa**b**”,”az**b**”)

↓

1+ LCS(“aa”,”az”)

How to decompose the original problem?

LCS Intuitive Solution – Dynamic Programming

Longest Common Subsequence

X “aab”
Y “azb”

LCS(“aa**b**”,”az**b**”)

1+ LCS(“a**a**”,”a**z**”)

max(LCS(“**a**”,”a**z**”), LCS(“a**a**”,”**a**”))

max(LCS(“_”,”a**z**”), LCS(“**a**”,”**a**”))

1+ LCS(“**a**”,”_”),

1+ LCS(“_”,”_”),

LCS DP Algorithm

```
LCS-LENGTH(X, Y)
m ← length[X]
n ← length[Y]
for i ← 1 to m
      do c[i, 0] ← 0
for j ← 0 to n
      do c[0, j] ← 0
for i ← 1 to m
      do for j ← 1 to n
              do if x_i = y_j
                    then c[i, j] ← c[i − 1, j − 1] + 1
                         b[i, j] ← "↖"
                    else if c[i − 1, j] ≥ c[i, j − 1]
                            then c[i, j] ← c[i − 1, j]
                                 b[i, j] ← "↑"
                            else c[i, j] ← c[i, j − 1]
                                 b[i, j] ← "←"
return c and b
```

LCS DP – Step 3: Computing the Length of LCS

- *Matrix* $c[0..m,0..n]$, where $c[i,j]$ is defined as above.
  - $c[m,n]$ is the answer (length of LCS).
- $b[1..m,1..n]$, where $b[i,j]$ points to the table entry corresponding to the optimal subproblem solution chosen when computing $c[i,j]$.
  - From $b[m,n]$ backward to find the LCS.

LCS Intuitive Solution – Dynamic Programming

Longest Common Subsequence

X “aab”
Y “azb”

		a	a	b
a				
z				
b				

LCS Intuitive Solution – Dynamic Programming

Longest Common Subsequence

X “aab”
Y “azb”

		a	a	b
	0	0	0	0
a	0	1 ↖	1 ↖	1 ←
z	0	1 ↑	1 ↑	1 ↑
b	0	1 ↑	1 ↑	2 ↖

LCS DP – Step 4: Constructing LCS

```
PRINT-LCS(b, X, i, j)
if i = 0 or j = 0
    then return
if b[i, j] = "↖"
    then PRINT-LCS(b, X, i − 1, j − 1)
            print x_i
elseif b[i, j] = "↑"
    then PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

LCS Intuitive Solution – Dynamic Programming

Longest Common Subsequence

X “aab”
Y “azb”

```
PRINT-LCS(b, X, i, j)
if i = 0 or j = 0
    then return
if b[i, j] = "↖"
    then PRINT-LCS(b, X, i − 1, j − 1)
         print x_i
elseif b[i, j] = "↑"
    then PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

		a	a	b
	0	0	0	0
a	0	1 ↖	1 ↖	1 ←
z	0	1 ↑	1 ↑	1 ↑
b	0	1 ↑	1 ↑	2 ↖

i	j	0	1	2	3	4	5	6
	y_j		B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

Figure 15.6 The c and b tables computed by LCS-LENGTH on the sequences $X = \langle A, B, C, B, D, A, B \rangle$ and $Y = \langle B, D, C, A, B, A \rangle$. The square in row i and column j contains the value of $c[i, j]$ and the appropriate arrow for the value of $b[i, j]$. The entry 4 in $c[7, 6]$—the lower right-hand corner of the table—is the length of an LCS $\langle B, C, B, A \rangle$ of X and Y. For $i, j > 0$, entry $c[i, j]$ depends only on whether $x_i = y_j$ and the values in entries $c[i-1, j]$, $c[i, j-1]$, and $c[i-1, j-1]$, which are computed before $c[i, j]$. To reconstruct the elements of an LCS, follow the $b[i, j]$ arrows from the lower right-hand corner; the path is shaded. Each "↖" on the path corresponds to an entry (highlighted) for which $x_i = y_j$ is a member of an LCS.

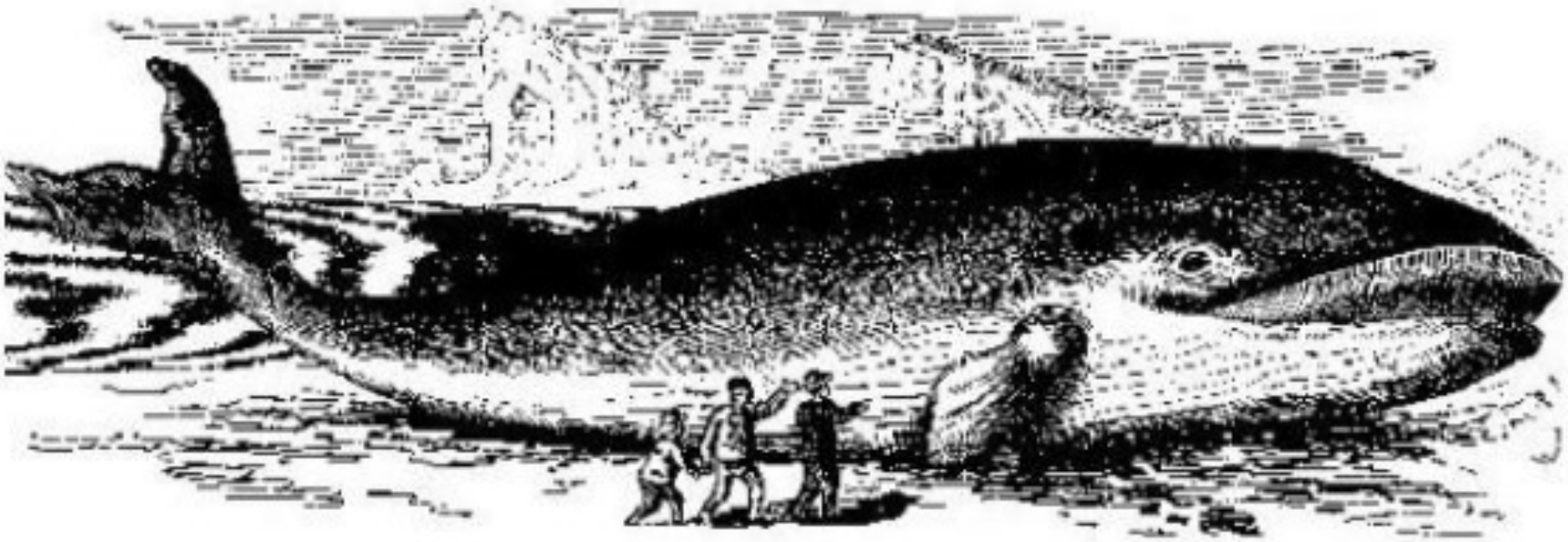
LCS Intuitive Solution – Dynamic Programming

Longest Common Subsequence: Exercise

X “acb”
Y “czb”

		a	c	b
c				
z				
b				

Sequence Alignment



```
WHALE   GTGTGGTCTCGTGATCAAAGGCGAAAGGTGGCTCTAGAGAATCCC
        ||||||||||| ||||| ||||| ||| ||||||||||||||||||
HUMAN   GTGTGGTCTCGCGATCAGAGGCGCAAGATGGCTCTAGAGAATCCC
```

Pairwise Sequence Alignment Problem

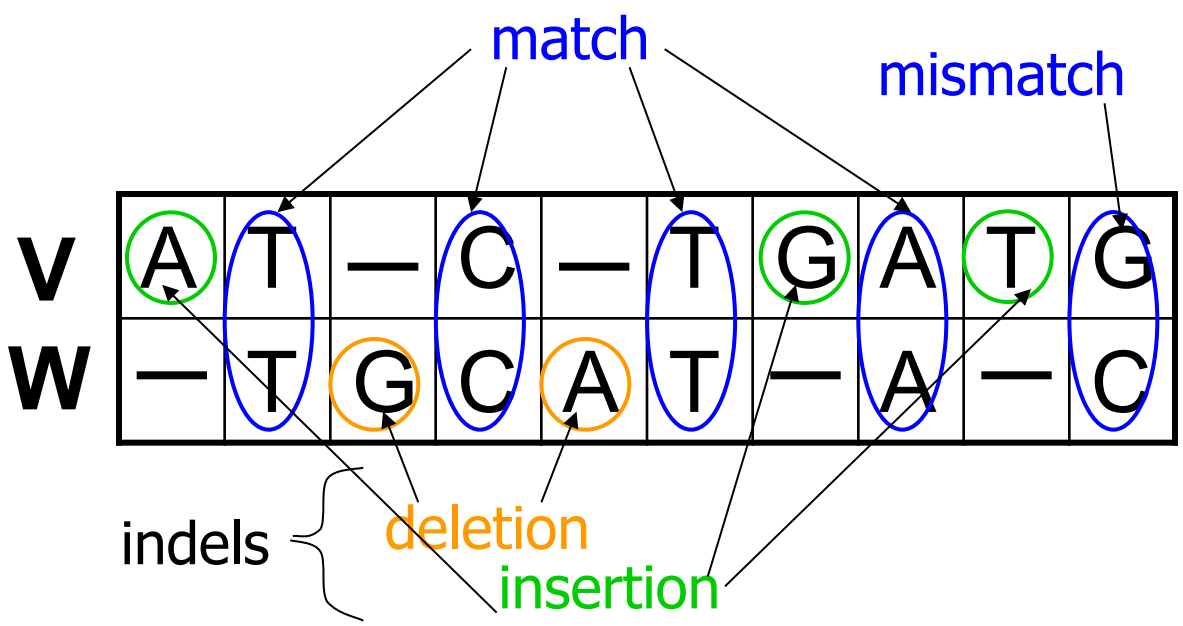
Aligning DNA Sequences

Alignment : 2 x ***k*** matrix (***k*** ≥ ***m***, ***n***)

v = ATCTGATG n = 8

w = TGCATAC m = 7

4 matches
1 mismatches
3 insertions
2 deletions



Pairwise Sequence Alignment Problem

- Given
  - a pair of sequences (DNA or protein)
  - a method for scoring a candidate alignment
- Do
  - find an alignment for which the score is maximized

What do we mean by this?

TREE V.S. REED

TREE REED	TREE_ _REED	TRE_E_ _REED	TRE_E_ _RE__ ED	…

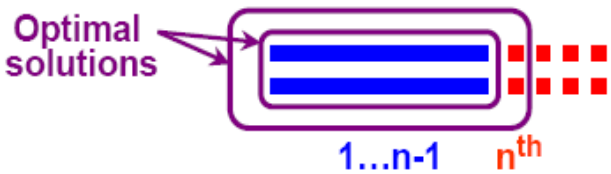
Difficulty of Pairwise Sequence Alignment (PSA)

- Consider two sequences of length *n*
  - Number of possible global alignments

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx \frac{2^{2n}}{\sqrt{(\pi n)}}$$

  - For n = 40, there are over 10^{23} possible alignments
- Approaches
  - "Optimal" solution
    - Dynamic programming
  - Heuristic solutions
    - Dot matrix plot
    - FASTA
    - BLAST

Dynamic Programming (DP)



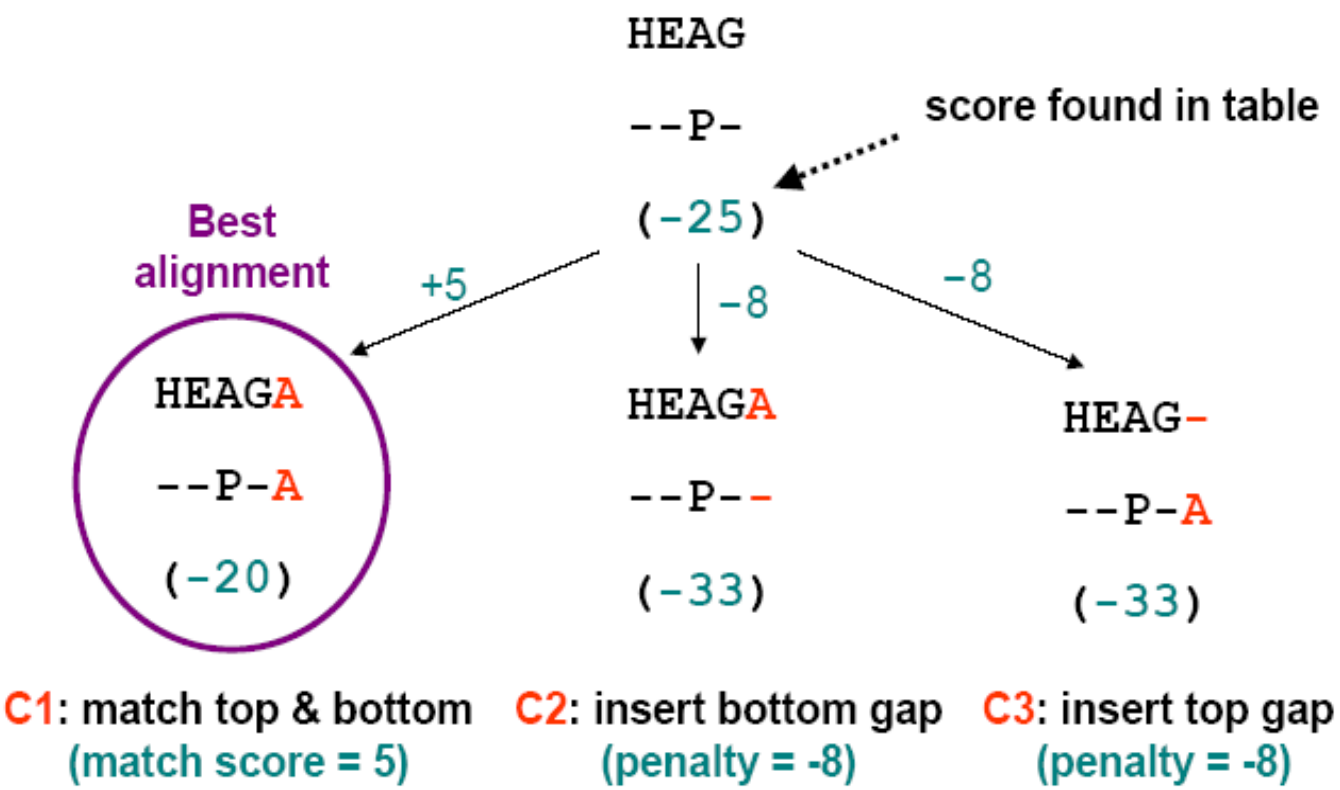
- Alignment
  - Problem can be subdivided
  - Optimal alignment of n bases
    - 1. Incorporates optimal alignment of 1…n-1 bases
    - 2. Combine with best alignment for n^{th} base
- Bioinformatics application
    - Global alignment [Needleman-Wunsch 1970]
    - Local alignment [Smith-Waterman 1981]
- Complexity
    - $O(n^2)$ or $O(n \times m)$ algorithm (sequences of length n, m)
    - Feasible for moderate-sized sequences, not entire genomes

Dynamic Programming (DP)

- Dynamic programming usually consists of three components.
  - Recursive relation (Initialization)
  - Tabular computation (Matrix Filling)
  - Trace back (Trace Back)
- This efficient recursive method is used to search through all possible alignments and find the one with the optimal score.

Intuition of Dynamic Programming

Considering sequences: “HEAGA” and “PA”



Intuition of Dynamic Programming

If we already have the optimal solution to:

XY
AB

then we know the **next** pair of characters will either be:

XYZ	**or**	**XY-**	**or**	**XYZ**
ABC		**ABC**		**AB-**

(where "-" indicates a gap).

So we can extend the match by determining which of these has the highest score.

DP (Global Alignment)

- **Recursive formula**

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j) & \text{c1} \\ F(i-1,j) + d & \text{c2} \\ F(i,j-1) + d & \text{c3} \end{cases}$$

- Notation
  - x_i — i^{th} letter of string x
  - y_j — j^{th} letter of string y
  - $x_{1..i}$ — prefix of x from letters 1 through i
  - F — matrix of optimal scores (DP Matrix)
    F(i, j) represents optimal score lining up $x_{1..i}$ with $y_{1..j}$
  - d — gap penalty
  - s — scoring matrix

Constant vs. Non-constant Gap Penalty

- Constant gap penalty

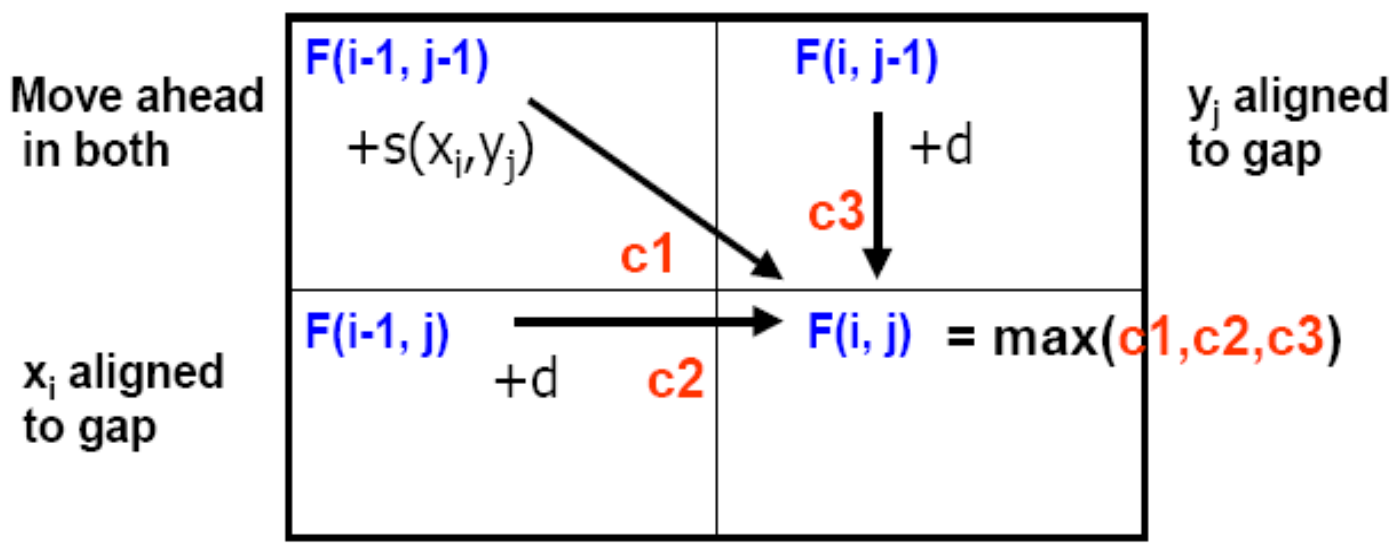
$$F(i,j) = \max\begin{cases} F(i-1,j-1) + s(x_i, y_j), \\ F(i-1,j) + d, \\ F(i,j-1) + d. \end{cases}$$

- Non-constant gap penalty

$$F(i,j) = \max\begin{cases} F(i-1,j-1) + s(x_i, y_j), \\ F(k,j) + \gamma(i-k), \quad k = 0,\ldots,i-1, \\ F(i,k) + \gamma(j-k), \quad k = 0,\ldots,j-1. \end{cases}$$

DP (Global Alignment)

- Algorithm
  - Initialize: F(0,0) = 0, F(i,0) = i $\times$ d, F(0, j) = j $\times$ d *(gap penalties)*
  - Fill from top left to bottom right using recursive formula



While building the table, keep track of where optimal score came from, then reverse arrows

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j) & \text{c1} \\ F(i-1,j) + d & \text{c2} \\ F(i,j-1) + d & \text{c3} \end{cases}$$

DP Example

- Sequences
  - Seq1: HEAGAWGHEE
  - Seq2: PAWHEAE
  - Gap Penalty: -8 (Constant)
  - Scoring Matrix (Blosum 50)

	H	E	A	G	A	W	G	H	E	E
P	-2	-1	-1	-2	-1	-4	-2	-2	-1	-1
A	-2	-1	5	0	5	-3	0	-2	-1	-1
W	-3	-3	-3	-3	-3	15	-3	-3	-3	-3
H	10	0	-2	-2	-2	-3	-2	10	0	0
E	0	6	-1	-3	-1	-3	-3	0	6	6
A	-2	-1	5	0	5	-3	0	-2	-1	-1
E	0	6	-1	-3	-1	-3	-3	0	6	6

DP Example

Algorithm

Initialize
F(0,0) = 0, F(i,0) = i × d, F(0, j) = j × d
(gap penalties)

Fill from top left to bottom right using recursive formula

		H	E	A	G	A	W	G	H	E	E
	0	-8	-16	-24	-32	-40	-48	-56	-64	-72	-80
P	-8	-2	-9	-17	-25	-33	-42	-49	-57	-65	-73
A	-16										
W	-24										
H	-32										
E	-40										
A	-48										
E	-56										

Blosum 50

	H	E	A	G	A	W	G	H	E	E
P	-2	-1	-1	-2	-1	-4	-2	-2	-1	-1
A	-2	-1	5	0	5	-3	0	-2	-1	-1
W	-3	-3	-3	-3	-3	15	-3	-3	-3	-3
H	10	0	-2	-2	-2	-3	-2	10	0	0
E	0	6	-1	-3	-1	-3	-3	0	6	6
A	-2	-1	5	0	5	-3	0	-2	-1	-1
E	0	6	-1	-3	-1	-3	-3	0	6	6

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j) & \text{c1} \\ F(i-1,j) + d & \text{c2} \\ F(i,j-1) + d & \text{c3} \end{cases}$$

DP Example

Algorithm

Initialize
F(0,0) = 0, F(i,0) = i × d, F(0, j) = j × d
(gap penalties)

Fill from top left to bottom right using recursive formula

		H	E	A	G	A	W	G	H	E	E
	0	-8	-16	-24	-32	-40	-48	-56	-64	-72	-80
P	-8	-2	-9	-17	-25	-33	-42	-49	-57	-65	-73
A	-16	-10	-3	-4	-12	-20	-28	-36	-44	-52	-60
W	-24	-18	-11	-6	-7	-15	-5	-13	-21	-29	-37
H	-32	-14	-18	-13	-8	-9	-13	-7	-3	-11	-19
E	-40	-22	-8	-16	-16	-9	-12	-15	-7	3	-5
A	-48	-30	-16	-3	-11	-11	-12	-12	-15	-5	2
E	-56	-38	-24	-11	-6	-12	-14	-15	-12	-9	1

Blosum 50

	H	E	A	G	A	W	G	H	E	E
P	-2	-1	-1	-2	-1	-4	-2	-2	-1	-1
A	-2	-1	5	0	5	-3	0	-2	-1	-1
W	-3	-3	-3	-3	-3	15	-3	-3	-3	-3
H	10	0	-2	-2	-2	-3	-2	10	0	0
E	0	6	-1	-3	-1	-3	-3	0	6	6
A	-2	-1	5	0	5	-3	0	-2	-1	-1
E	0	6	-1	-3	-1	-3	-3	0	6	6

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j) & \text{c1} \\ F(i-1,j) + d & \text{c2} \\ F(i,j-1) + d & \text{c3} \end{cases}$$

DP Example (Traceback)

- Trace arrows from bottom right to top left
  - Diagonal – both match
  - Up – left sequence match a gap
    - Or insert a gap to top sequence
  - Left – top sequence match a gap
    - Or insert a gap to left sequence

		H	E	A	G	A	W	G	H	E	E
	0	-8	-16	-24	-32	-40	-48	-56	-64	-72	-80
P	-8	-2	-9	-17	-25	-33	-42	-49	-57	-65	-73
A	-16	-10	-3	-4	-12	-20	-28	-36	-44	-52	-60
W	-24	-18	-11	-6	-7	-15	-5	-13	-21	-29	-37
H	-32	-14	-18	-13	-8	-9	-13	-7	-3	-11	-19
E	-40	-22	-8	-16	-16	-9	-12	-15	-7	3	-5
A	-48	-30	-16	-3	-11	-11	-12	-12	-15	-5	2
E	-56	-38	-24	-11	-6	-12	-14	-15	-12	-9	1

Optimal global alignment

```
HEAGAWGHE-E
--P-AW-HEAE
```

Global alignment vs. local alignment

- **Global alignment**: the entire sequence of each protein or DNA sequence is contained in the alignment.
- **Local alignment**: only regions of greatest similarity between two sequences are aligned

```
  1 MKWVWALLLLAAWAAAERDCRVSSFRVKENFDKARFSGTWYAMAKKDPEG 50
      .  |||  |      .   |.  .   .  |  :  .|||||.:|    :
  1 ...MKCLLLALALTCGAQALIVT..QTMKGLDIQKVAGTWYSLAMAASD. 44

 51 LFLQDNIVAEFSVDETGQMSATAKGRVR.LLNNWD..VCADMVGTFTDTE 97
    : | |    |   |     ::  | .| .  ||  |:   ||        |.
 45 ISLLDAQSAPLRV.YVEELKPTPEGDLEILLQKWENGECAQKKIIAEKTK 93

 98 DPAKFKMKYWGVASFLQKGNDDHWIVDTDYDTYAV...........QYSC 136
     || ||.         |         :.|||||  | .          .|
 94 IPAVFKIDALNENKVL........VLDTDYKKYLLFCMENSAEPEQSLAC 135

137 RLLNLDGTCADSYSFVFSRDPNGLPPEAQKIVRQRQ.EELCLARQYRLIV 185
    . |        |     | :    ||     .        | || |
136 QCLVRTPEVDDEALEKFDKALKALPMHIRLSFNPTQLEEQCHI....... 178
                                .
```

percent identity: ~26%

```
RBP:          26  RVKENFDKARFSGTWYAMAKKDPEGLFLQDNIVAEFSVDETGQMSATAKGRVRLLNNWD- 84
                  + K++ +  + +GTW++MA      + L   + A   V  T  +        +L+  W+
glycodelin:   23  QTKQDLELPKLAGTWHSMAMA-TNNISLMATLKAPLRVHITSLLPTPEDNLEIVLHRWEN 81
```

Global alignment vs. local alignment

- Global alignment are often not effective for highly diverged sequences - do not reflect the biological reality that two sequences may only share limited regions of conserved sequence.
- Global methods are useful when you want to force two sequences to align over their entire length
- Local alignment is almost always used for database searches such as BLAST. It is useful to find domains (or limited regions of homology) within sequences.

DP (Local Alignment)

- Make 0 minimal score (i.e., start new alignment)
- Alignment can start / end anywhere
  - Start at highest score(s)
  - End when 0 reached

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j) \\ F(i-1,j) + d \\ F(i,j-1) + d \\ 0 \end{cases}$$

DP (Local Alignment)

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j) \\ F(i-1,j) - 8 \\ F(i,j-1) - 8 \\ 0 \end{cases}$$

		H	E	A	G	A	W	G	H	E	E
	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	5	0	5	0	0	0	0	0
W	0	0	0	0	2	0	20	12	4	0	0
H	0	10	2	0	0	0	12	18	22	14	6
E	0	2	16	8	0	0	4	10	18	28	20
A	0	0	8	21	13	5	0	4	10	20	27
E	0	0	6	13	18	12	4	0	4	16	26

Blosum 50

	H	E	A	G	A	W	G	H	E	E
P	-2	-1	-1	-2	-1	-4	-2	-2	-1	-1
A	-2	-1	5	0	5	-3	0	-2	-1	-1
W	-3	-3	-3	-3	-3	15	-3	-3	-3	-3
H	10	0	-2	-2	-2	-3	-2	10	0	0
E	0	6	-1	-3	-1	-3	-3	0	6	6
A	-2	-1	5	0	5	-3	0	-2	-1	-1
E	0	6	-1	-3	-1	-3	-3	0	6	6

DP (Local Alignment)

- Traceback
  - Start at highest score and trace arrows back to first 0

		H	E	A	G	A	W	G	H	E	E
	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	5	0	5	0	0	0	0	0
W	0	0	0	0	2	0	20	12	4	0	0
H	0	10	2	0	0	0	12	18	22	14	6
E	0	2	16	8	0	0	4	10	18	28	20
A	0	0	8	21	13	5	0	4	10	20	27
E	0	0	6	13	18	12	4	0	4	16	26

Highest score

Optimal local alignment

```
AWGHE
AW-HE
```

Learning Outcomes

- Longest Common Subsequence (LCS)
- Pairwise Sequence Alignment Problem
  1. Global Alignment
  2. Local Alignment