BIO 285/CSCI 285/MATH 285
Bioinformatics
Programming Lecture 10
Pairwise Sequence Alignment 4
Dynamic Programming

Instructor: Lei Qian
Fisk University

Dynamic Programming

* Dynamic Programming can be used for both global and local alignment.
* Both global and local types of alignments may be made by simple changes in the basic DP algorithm.
* Alignments depend on the choice of a scoring system for comparing character pairs and penalty scores (e.g. PAM and BLOSUM matrixes – covered before)

Scoring functions – example:

w (match) = +2 or substitution matrix

w (mismatch) = -1 or substitution matrix

w (gap) = -3

Dynamic Programming

* **Global Alignment (Needleman-Wunsch)**

a) General goal is to obtain optimal global alignment between two sequences, allowing gaps.

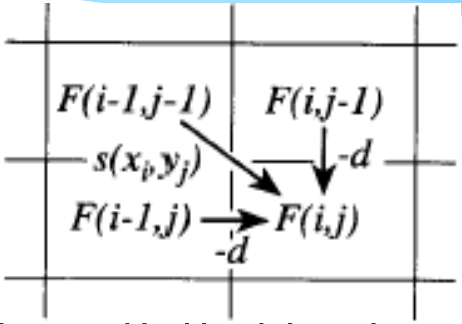
b) We construct a matrix *F* indexed by *i* and j, one index for each sequence, where the value F(*i*,j) is the score of the best alignment between the initial segment $x_1, x_2, \ldots, x_i$ and the initial segment $y_1, y_2, \ldots, y_j$. We begin by initializing F(0,0) = 0.
We then proceed to fill the matrix from top left to bottom right. If F(*i*-1, *j*-1), F(*i*-1,j) and F(*i*,*j*-1) are known, it is possible to calculate F(*i*,j).

Dynamic Programming

* **Global Alignment (Needleman-Wunsch)**

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

where $s(a,b)$ is the likelihood score that residues *a* and *b* occur as an aligned pair, and *d* is the gap penalty.



Once you construct the matrix, you trace back the path that leads to $F(n,m)$, which is by definition the **best score for an alignment** of $x_1, \ldots, x_n$ to $y_1, \ldots, y_m$.

Dynamic Programming

- **Global Alignment (Example)**

  Gap (delete or insert) penalty = -8

	λ	H	E	A	G	A	W	G	H	E	E
λ	0	-8	-16	-24	-32	-40	-48	-56	-64	-72	-80
P	-8	-2	-9	-17	-25	-33	-41	-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

```
HEAGAWGHE-E
--PA-W-HEAE
```

Dynamic Programming

Global Alignment Code

```
Assume we align two sequences s1 and s2 with length m and n.
We use Numpy to handle matrices.
import numpy as np

0. Define scores
char_to_int = {'A':0, 'C':1, 'G':2, 'T':3}
#convert characters to proper indices
indel = -2
#penalty for a gap
scoring = np.array([[2,-1,-1,-1],
                    [-1,2,-1,-1],
                    [-1,-1,2,-1],
                    [-1,-1,-1,2]])
```

Dynamic Programming

Global Alignment Code

Assume we align two sequences s1 and s2 with length m and n.

1. **Define matrices F with m+1 rows and n+1 columns**

```
m = len(seq1)
n = len(seq2)
F = np.zeros((m+1, n+1))
#Create an all zero matrix with m+1 rows and n+1 columns.
```

2. **Set values of the first row and the first column:**

```
for i in range(m+1):
    F[i,0] = i*indel
for j in range(n+1):
    F[0,j] = j*indel
```

Dynamic Programming

Global Alignment Code

Assume we align two sequences s1 and s2 with length m and n.

3. Calculate other values of the matrix:

```
 for i in range(1, m+1):
        for j in range(1, n+1):
            F[i,j] = max(F[i-1, j-1] + score(seq1[i-1], seq2[j-1]),
                F[i-1, j] + indel,
                F[i, j-1] + indel)
```

Dynamic Programming

Global Alignment Code

Assume we align two sequences s1 and s2 with length m and n.

4. Trace back the matrix to find the path:

```
def get_aligned_pair(seq1, seq2, i, j):
    if i==0:
        c1 = '-'
    else:
        c1 = seq1[i-1]
    if j==0:
        c2 = '-'
    else:
        c2 = seq1[j-1]
    return (c1, c2)
```

Dynamic Programming

Global Alignment Code

Assume we align two sequences s1 and s2 with length m and n.

4. Trace back the matrix to find the path:

```
alignment= []
i = len(seq1)
j = len(seq2) #(i, j) is the location of the bottom right corner
while i>0 and j>0:
    if F[i-1, j-1] + score(seq1[i-1], seq2[j-1]) == F[i,j]:
        alignment.append(get_aligned_pair(seq1, seq2, i, j))
        i -= 1
        j -= 1
    elif F[i-1, j] + indel == F[i,j]:
        alignment.append(get_aligned_pair(seq1, seq2, i, 0))
        i -= 1
    else:
        alignment.append(get_aligned_pair(seq1, seq2, 0, j))
        j -= 1
```

Dynamic Programming

Global Alignment Code

Assume we align two sequences s1 and s2 with length m and n.

4. Trace back the matrix to find the path:

```
    #in case we trace back to the first row or first column but not the
    #top left corner
    while i > 0:
        alignment.append(get_aligned_pair(seq1, seq2, i, 0))
        i -= 1
    while j > 0:
        alignment.append(get_aligned_pair(seq1, seq2, 0, j))
        j -= 1

     #reverse the trace back path
     alignment.reverse()
```

Dynamic Programming

Global Alignment Code

Assume we align two sequences s1 and s2 with length m and n.

5. Print the alignment:

```
def print_sequences(pairs):
    top_seq = []
    bottom_seq = []
    for (b, t) in pairs:
        bottom_seq.append(b)
        top_seq.append(t)
    for n in top_seq:
        print n,
    print ' '
    for n in bottom_seq:
        print n,
```

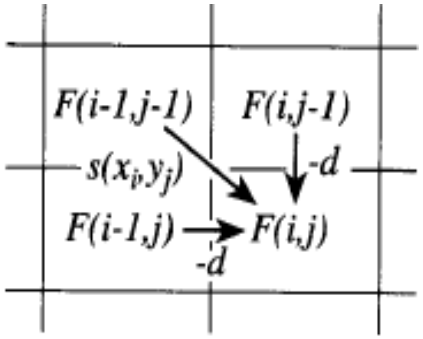
Dynamic Programming

Local alignment (Smith-Waterman)

Two changes from global alignment:

1. Possibility of taking the value 0 if all other options have value less than 0. This corresponds to starting a new alignment.

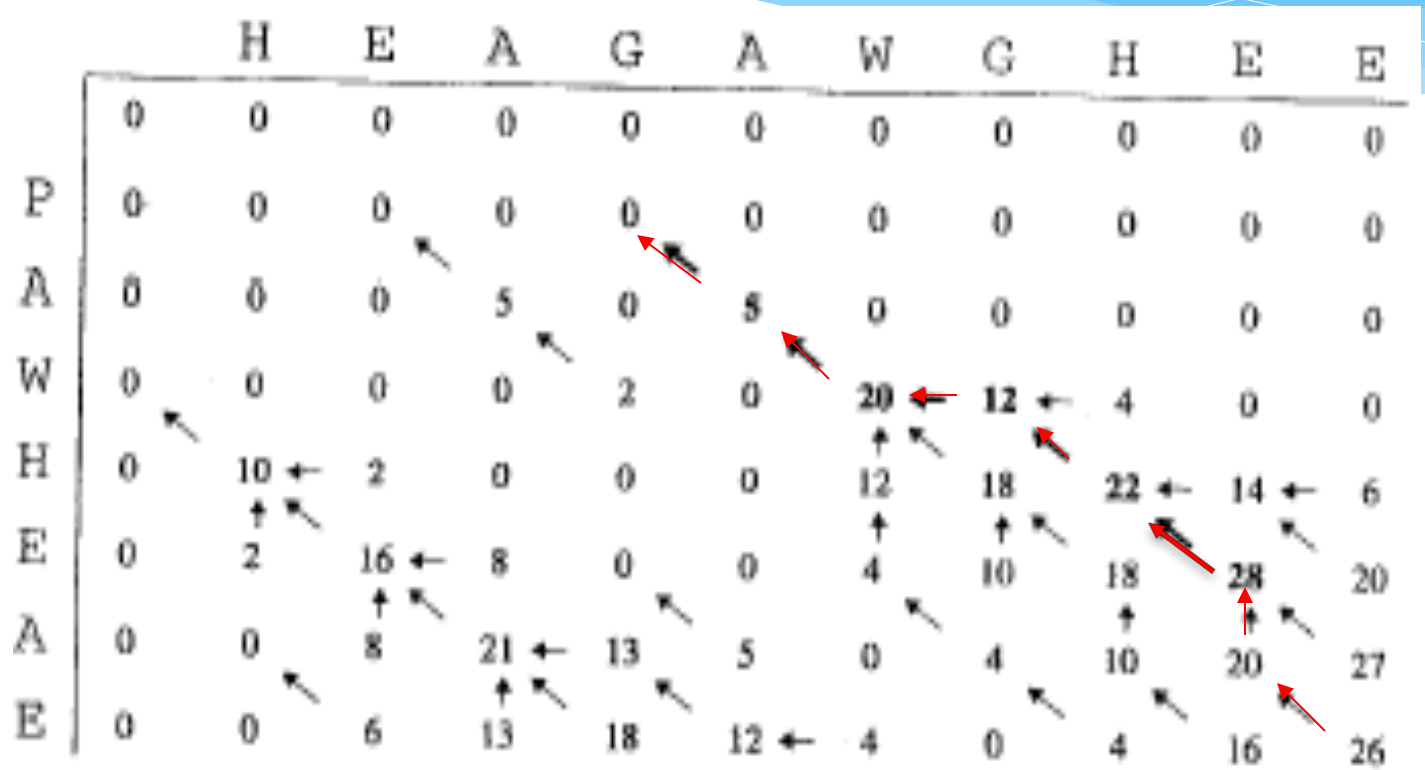
2. Alignments can end anywhere in the matrix, so instead of taking the value in the bottom right corner, F(n,m) for the best score, we look for the highest value of F(i,j) over the whole matrix and start the trace-back from there.



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

Dynamic Programming

Local alignment (Smith-Waterman)



		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

AWGHE-E

AW-HEAE

Dynamic Programming

Dynamic Programming

- **Advantages:**
  Guaranteed in a mathematical sense to provide the optimal (very best or highest-scoring) alignment for a given set of scoring functions.

- **Disadvantages:**

a) Slow due to the very large number of computational steps: $O(n^2)$.

b) Computer memory requirement also increases as the square of the sequence lengths.

Therefore, it is difficult to use the method for very long sequences.