

Combinatorial Methods in Bioinformatics

Essence of Sequence Comparison

Wing-Kin Sung, Ken 宋永健

ksung@comp.nus.edu.sg

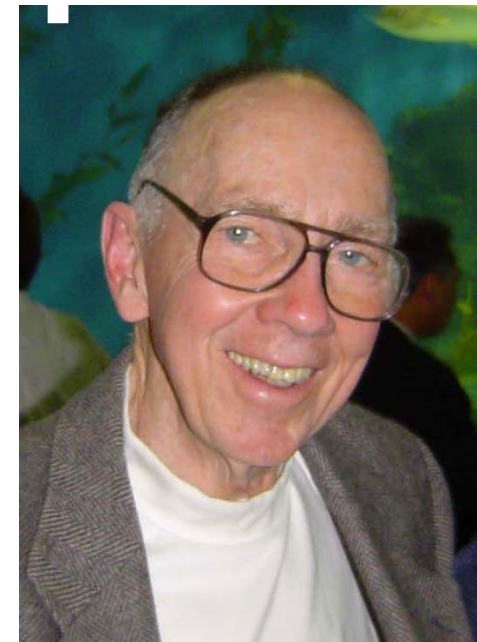
Earliest Researches in Sequence Comparison

- Doolittle et al. (Science, July 1983) searched for platelet-derived growth factor (PDGF) in his own DB. He found that PDGF is similar to v-sis onc gene.

```
• PDGF-2 1 SLGSLTIAEPAMIAECKTREEVFCICRRL?DR?? 34
  p28sis 61 LARGKRSLGSLSVAEPAMIAECKTRTEVFEISRRLIDRTN 100
```

- Riordan et al. (Science, Sept 1989) wanted to understand the cystic fibrosis gene:

```
CFTR (N) FSLLGTPVLKDINFKIERGQI-LAVAGSTGAGKTSLLMMIMG
CFTR (C) YTEGGNAILENISFSISPGQRVGLLGRGTSGSKSTLLSAFLR
hmdr1 (N) PSRKEVKILKGLNLKVQSGQTVALVGNSSGCGKSTTVQLMQR
hmdr1 (C) PTRPDIPVLQGLSLEVKKGQTLALVGSSGCGKSTTVQLLER
mmdr1 (N) PSRSEVQILKGLNLKVKSGQTVALVGNSSGCGKSTTVQLMQR
mmdr1 (C) PTRPNIPVLQGLSLEVKKGQTLALVGSSGCGKSTTVQLLER
mmdr2 (N) PSRANIKILKGLNLKVKSGQTVALVGNSSGCGKSTTVQLLQR
mmdr2 (C) PTRANVPVLQGLSLEVKKGQTLALVGSSGCGKSTTVQLLER
pfmdr (N) DTRKDVEIYKDLSTLLKEGKTYAFVGESGCGKSTILKLE
pfmdr (C) ISRPNVPIYKNLSFTCDSSKTTAIVGETGSGKSTFMNLLLR
STE6 (N) PSRPSEAVLKNVSLNFSAGQFTFIVGKSGSGKSTLSNLLLR
STE6 (C) PSAPTAFFVYKNMNFDMFCGQTLGIIGESGTGKSTLVLLTK
hlyB YKPDSPVILDNINISIKQGEVIGIVGRSGSGKSTLIKLIQR
White IPAPRKHLLKNVCGVAYPGELLAVMGSSGAGKTLLNALAF
MbpX KSLGNLKIILDRVSLYVPKFSLIALLGPSGSGKSSLLRILAG
BtuD QDVAESTRLGPLSGEVVRAGRILHLVGPNGAGKSTLLARIAG
```



Russell Doolittle

Why we need to compare sequences?

- Biology has the following conjecture
 - Given two DNAs (or RNAs, or Proteins), high similarity → similar function or similar 3D structure
- Thus, in bioinformatics, we always compare the similarity of two biological sequences.

String edit problem

- String edit minimizes the number of edit operations to convert a sequence to another.
- We can associate a **cost** to each operation.
- Our aim is to minimize the total cost. Such cost is called **edit distance**.
- The edit distance of the following example is 3.

insert → S = A CAATCC
 T = A GCA TGC
 0 1 0 0 1 0 1 0
 delete ← mismatch

match

$$\delta(C, G) = -1$$

	_	A	C	G	T
_		1	1	1	1
A	1	0	1	1	1
C	1	1	0	1	1
G	1	1	1	0	1
T	1	1	1	1	0

Cost function

String alignment problem (Global alignment problem)

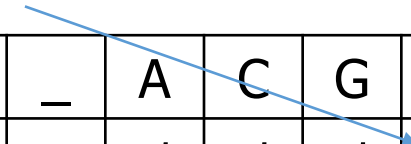
- The **string alignment problem** aims to find an alignment that maximizes the total similarity scores of all the aligned pairs.
- Consider two strings ACAATCC and AGCATGC. $\delta(_,T) = -1$

- One of their **alignment** is

S= A _ CAATCC

T= AGCA _ TGC

- The alignment has similarity score 7
- In the above alignment,
 - space ('_') is introduced to both strings
 - There are 5 matches, 1 mismatch, 1 insert, and 1 delete.



	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

Similarity function

String alignment problem (Global alignment problem)

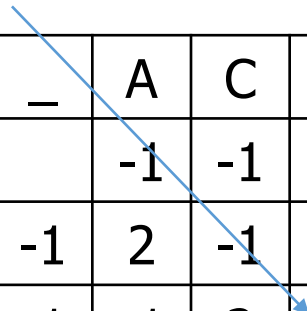
- The **string alignment problem** aims to find an alignment that maximizes the total similarity scores of all the aligned pairs.
- Consider two strings ACAATCC and AGCATGC. $\delta(C,G) = -1$

- One of their **alignment** is

S= A _ CAATCC

T= AGCA _ TGC

- The alignment has similarity score 7
- Note that the above alignment has the maximum score. Such alignment is called **optimal alignment**.



	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

Similarity function

Similarity vs. Distance

- String edit problem for a cost function C is equivalent to String alignment problem for a similarity function $-C$.
- In this lecture, we only study string alignment!

	_	A	C	G	T
_		1	1	1	1
A	1	-2	1	1	1
C	1	1	-2	1	1
G	1	1	1	-2	1
T	1	1	1	1	-2

Cost function

	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

Similarity function

Alignment:

Simple-Minded Probability & Score

Each aligned pair in an alignment is either: match, mismatch, insert and delete.

Model: p is the probability of match, q is the probability of mismatch, r is the probability of both insert and delete. (Note: $p+q+2r=1$)

Suppose an alignment A has m matches, n mismatches and h indels.

Then, the probability of the alignment A is

$$\text{prob}(A) = p^m \cdot q^n \cdot r^h$$

- For a random alignment, $p=q=r=0.25$. The probability of a random alignment is 0.25^{m+n+h} .
- Define score $S(A)$ by a simple log odds ratio as $\log \frac{\text{prob}(A)}{0.25^{m+n+h}}$.

String alignment problem

	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

- Input:
 - A similarity score matrix $\delta(x,y)$
 - Two sequences $S[1..n]$ and $T[1..m]$
- Output: An alignment of S and T that maximizes the alignment score $\text{Score}(S, T)$.
- Example: $S = \text{ACAATCC}$ and $T = \text{AGCATGC}$.
- The optimal alignment is

$$\begin{array}{ccccccc} & A & _ & C & A & T & C & C \\ & _ & A & G & C & A & _ & T & G & C \end{array}$$
- The optimal alignment score is $\text{Score}(S, T) = \delta(A,A) + \delta(_,G) + \delta(C,C) + \delta(A,A) + \delta(A,_) + \delta(T,T) + \delta(C,G) + \delta(C,C) = 7$.

How to align two sequences?

- A simple solution:
 - Generate all alignments of S and T.
 - Among all alignments, we report the alignment with the highest score.
- However, the number of possible alignments is at least $\binom{|S| + |T|}{|S|}$. (Can you prove it?)
- This solution runs in exponential time.
- (For example, to align two sequences of length 100, the time is at least 2^{100} CPU steps!)
- In computer science, there is a technique called dynamic programming.
- We can apply dynamic programming to align two sequences.

Dynamic programming (DP)

- Dynamic programming aims to find an optimal solution for a problem by solving a few subproblems.
- It involves three steps:
 1. Characterize the structure of the problem
 2. Find the recursive formula
 3. Solve the recursive formula by bottom-up dynamic programming
- Below, we illustrate how to use DP to align two sequences.

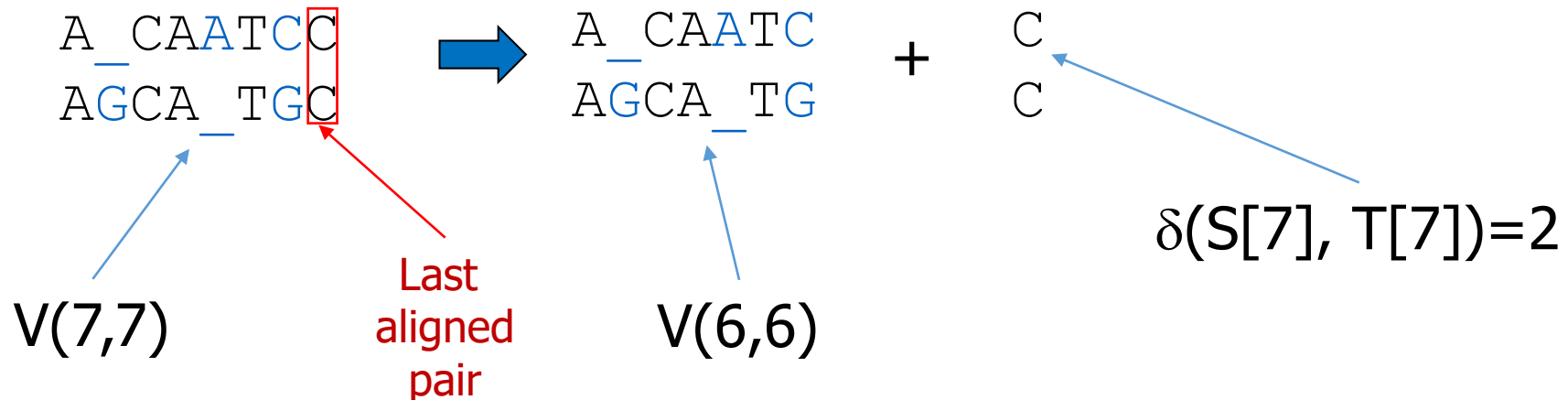
Characterize the problem

- Recall that $\text{Score}(S[1..n], T[1..m])$ is the optimal alignment score between $S[1..n]$ and $T[1..m]$

This observation implies that $V(n,m)$ can be solved if we know the solutions for $V(i,j)$ for $i < n$ and $j < m$.

- To simp
- We aim

- In other word, $V(7,7) = V(6,6) + \delta(S[7], T[7])$.



Find recursive formula

- Identify the base case and recursive case of $V(i, j)$

- Basis ($i=0$ or $j=0$):

- $V(0, 0) = 0$

- $V(0, j) = V(0, j-1) + \delta(_, T[j])$

- Insert j times

- $V(i, 0) = V(i-1, 0) + \delta(S[i], _)$

- Delete i times

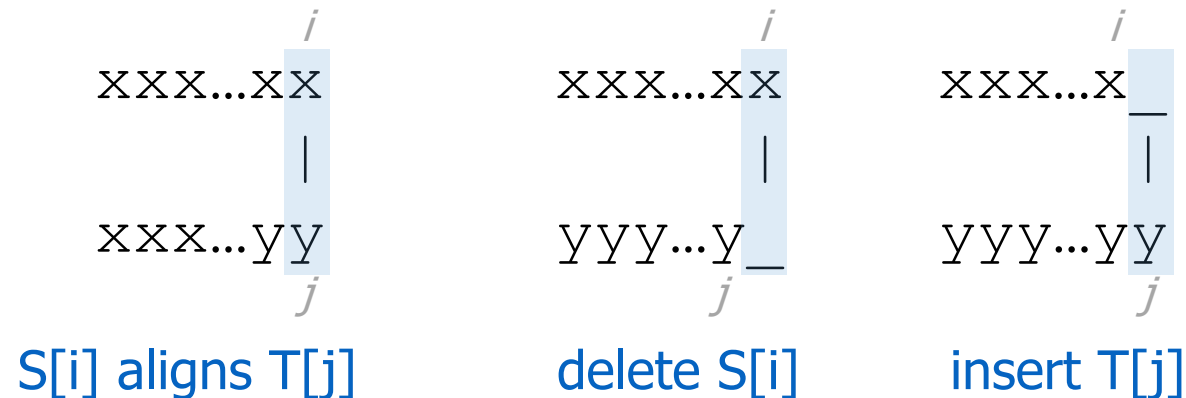
$$\begin{array}{lcl} \begin{array}{c} \text{---...--} \\ \text{yyy...yy} \end{array} & = & \begin{array}{c} \text{---...--} \\ \text{yyy...y} \end{array} + \begin{array}{c} \text{--} \\ \text{y} \end{array} \\ \begin{array}{c} \text{xxx...xx} \\ \text{---...--} \end{array} & = & \begin{array}{c} \text{xxx...x} \\ \text{---...--} \end{array} + \begin{array}{c} \text{x} \\ \text{--} \end{array} \end{array}$$

Find recursive formula

- Recurrence: For $i > 0, j > 0$

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{S[i] aligns T[j]} \\ V(i-1, j) + \delta(S[i], _) & \text{Delete S[i]} \\ V(i, j-1) + \delta(_, T[j]) & \text{Insert T[j]} \end{cases}$$

- In the alignment, the last pair must be either (1) S[i] aligns T[j], (2) delete S[i] or (3) insert T[j].



Solve the recursive formula by bottom-up dynamic programming

- We have the Needleman-Wunsch algorithm.

1. $V(0,0)=0;$
 2. For $j = 1$ to m
 - $V(0,j) = V(0,j-1)+\delta(_,T[j]);$
 3. For $i = 1$ to n
 - $V(i,0)=V(i-1,0)+\delta(S[i],_);$
 4. For $i = 1$ to n
 - For $j = 1$ to m
 - $V(i,j) = \max\{ V(i-1,j-1)+\delta(S[i],T[j]), V(i-1,j)+\delta(S[i],_), V(i,j-1)+\delta(_,T[j]) \};$
 5. Return $V(n,m);$
-
- Base case
- Recursive case

Example (I)

Find alignment of
ACAATCC and AGCATGC

- Step 1.
- Set $V(0,0)=0$

	_	A	G	C	A	T	G	C
_	0							
A								
C								
A								
A								
T								
C								
C								

	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

Example (I)

Find alignment of
ACAATCC and AGCATGC

- Step 2.
- $V(0,j)=V(0,j-1)-1$
- $V(0,1)=V(0,0)-1=-1$
- $V(0,2)=V(0,1)-1=-2$
- So on.

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A								
C								
A								
A								
T								
C								
C								

	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

Example (I)

Find alignment of
ACAATCC and AGCATGC

- Step 3.
- $V(i,0)=V(i-1,0)-1$
- $V(1,0)=V(0,0)-1=-1$
- $V(2,0)=V(1,0)-1=-2$
- So on.

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1							
C	-2							
A	-3							
A	-4							
T	-5							
C	-6							
C	-7							

	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

Example (II)

- Step 4.

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) \\ V(i-1, j) + \delta(S[i], _) \\ V(i, j-1) + \delta(_, T[j]) \end{cases}$$

$$V(2,3) = \max \begin{cases} V(1,2) + \delta(C, C) \\ V(1,3) + \delta(C, _) \\ V(2,2) + \delta(_, C) \end{cases} = 3$$

$$V(2,4) = \max \begin{cases} V(1,3) + \delta(C, A) \\ V(1,4) + \delta(C, _) \\ V(2,3) + \delta(_, A) \end{cases} = 2$$

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3	2			
A	-3							
A	-4							
T	-5							
C	-6							
C	-7							

Example (III)

- Step 4.

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) \\ V(i-1, j) + \delta(S[i], _) \\ V(i, j-1) + \delta(_, T[j]) \end{cases}$$

- Step 5.
- Report $V(7,7)$

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3	2	1	0	-1
A	-3	0	0	2	5	4	3	2
A	-4	-1	-1	1	4	4	3	2
T	-5	-2	-2	0	3	6	5	4
C	-6	-3	-3	0	2	5	5	7
C	-7	-4	-4	-1	1	4	4	7

Analysis

- We need to fill in all entries in the table with $n \times m$ matrix.
- Each entries can be computed in $O(1)$ time.
- Time complexity = $O(nm)$
- Space complexity = $O(nm)$

Recover the alignment

Given the V table, we can recover the alignment by calling `back_tracking(V, n, m)`.

- `back_tracking(V, i, j)`
 - $A = []$
 - While ($i \neq 0$ or $j \neq 0$)
 - If ($i=0$) or ($V(i,j) = V(i,j-1) + \delta(_, T[j])$) /* Case 1: insert $T[j]$ */
 - $A = (_, T[j]) \bullet A$
 - $j = j - 1$
 - else if ($j=0$) or ($V(i,j) = V(i-1,j) + \delta(S[i], _)$) /* Case 2: delete $S[i]$ */
 - $A = (S[i], _) \bullet A$
 - $i = i - 1$
 - else if ($V(i,j) = V(i-1,j-1) + \delta(S[i], T[j])$) /* Case 3: $S[i]$ aligns with $T[j]$ */
 - $A = (S[i], T[j]) \bullet A$
 - $i = i - 1; j = j - 1;$
 - Return A

Example (IV)

$V(7,7)$
 ↓ Case 3: report (C,C)
 $V(6,6)$
 ↓ Case 3: report (C,G)
 $V(5,5)$
 ↓ Case 3: report (T,T)
 $V(4,4)$
 ↓ Case 2: report (A,_)
 $V(3,4)$
 ↓ Case 3: report (A,A)
 $V(2,3)$
 ↓ Case 3: report (C,C)
 $V(1,2)$
 ↓ Case 1: report (_,G)
 $V(1,1)$
 ↓ Case 3: report (A,A)
 $V(0,0)$

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3	2	1	0	-1
A	-3	0	0	2	5	4	3	2
A	-4	-1	-1	1	4	4	3	2
T	-5	-2	-2	0	3	6	5	4
C	-6	-3	-3	0	2	5	5	7
C	-7	-4	-4	-1	1	4	4	7

A _ C A A T C C
 A G C A _ T G C

Problem on Speed (I)

- Aho, Hirschberg, Ullman 1976
 - If we can only compare whether two symbols are equal or not, the string alignment problem can be solved in $\Omega(nm)$ time.
- Hirschberg 1978
 - If symbols are ordered and can be compared, the string alignment problem can be solved in $\Omega(n \log n)$ time.
- Masek and Paterson 1980
 - Based on **Four-Russian's paradigm**, the string alignment problem can be solved in $O(nm/\log^2 n)$ time.

Problem on Speed (II)

- Let d be the total number of inserts and deletes.
 - $0 \leq d \leq n+m$
- If d is smaller than $n+m$, can we get a better algorithm? Yes!

A _ CAATCC
 AGCA _ TGC

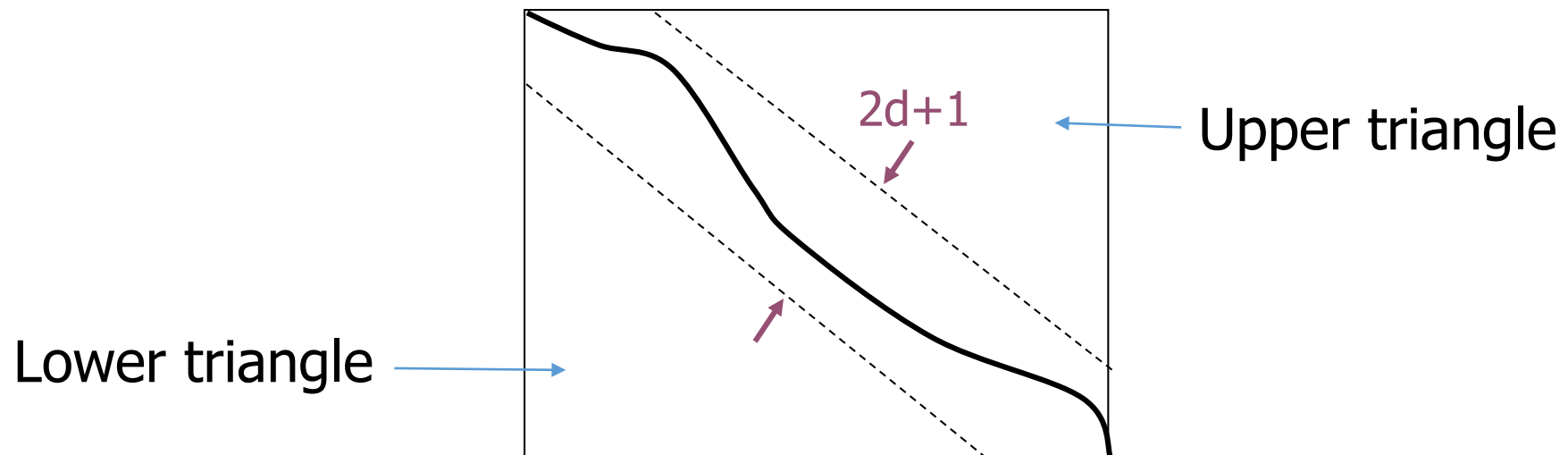
Diagonal and indel

- An entry $V(i,j)$ is on diagonal $(j-i)$.
- We observe that the back-tracking path change diagonal only when there is indel.
- For example,
 - when we move from $V(4,4)$ to $V(3,4)$, there is a deletion.
 - When we move from $V(1,2)$ to $V(1,1)$, there is an insertion.

	—	A	G	C	A	T	G	C
—	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3	2	1	0	-1
A	-3	0	0	2	5	4	3	2
A	-4	-1	-1	1	4	4	3	2
T	-5	-2	-2	0	3	6	5	4
C	-6	-3	-3	0	2	5	5	7
C	-7	-4	-4	-1	1	4	4	7

$O(dn)$ -time algorithm

- Observe that the alignment should be inside the $2d+1$ band.
- Thus, we don't need to fill-in the lower and upper triangle.
- Time complexity: $O(dn)$.



Example

• d=3

A _ C A A T C C

A G C A _ T G C

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3				
A	-1	2	1	0	-1			
C	-2	1	1	3	2	1		
A	-3	0	0	2	5	4	3	
A		-1	-1	1	4	4	3	2
T			-2	0	3	6	5	4
C				0	2	5	5	7
C					1	4	4	7

	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

Problem on Space

- Note that the dynamic programming requires a lot of space $O(mn)$.
- When we compare two very long sequences, space may be the limiting factor.
- Can we solve the string alignment problem in linear space?

Suppose we don't need to recover the alignment

- Observe that the V table can be filled in row by row by keeping two rows only.

- If we did not need to back track, space complexity = $O(\min(n, m))$

- Example:

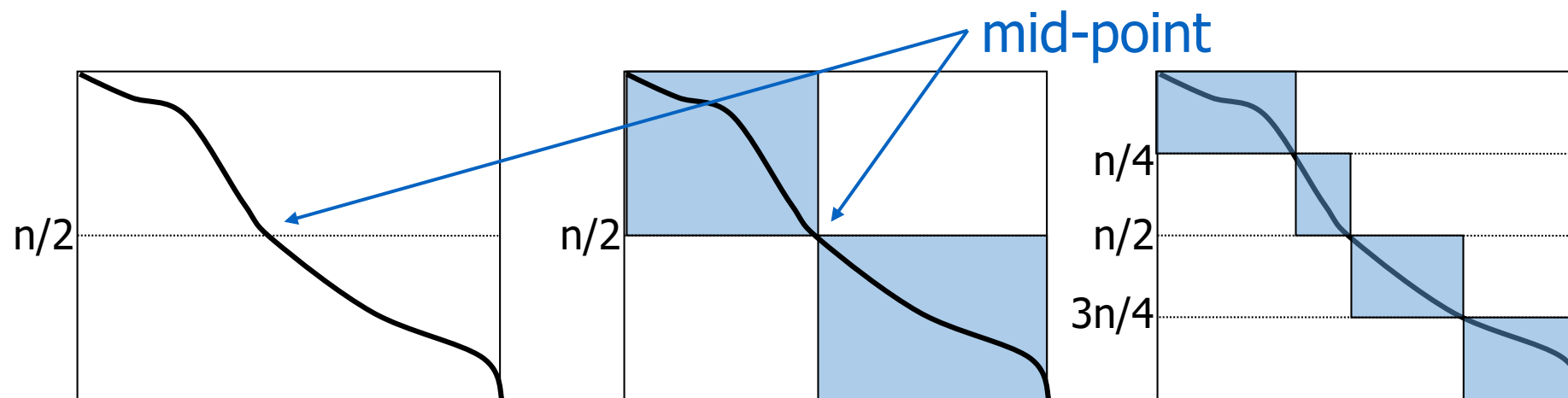
	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3	2	1	0	-1
A	-3	0	0	2	5	4	3	2
A	-4	-1	-1	1	4	4	3	2
T	-5	-2	-2	0	3	6	5	4
C	-6	-3	-3	0	2	5	5	7
C	-7	-4	-4	-1	1	4	4	7

- Note: when we fill in row 4, it only depends on row 3! So, we don't need to keep rows 1 and 2!
- Similarly, when we fill in row i , it only depends on row $i-1$! So, we only need to keep rows i and $i-1$.
- Thus, we only need to keep two rows during dynamic programming.
- We denote this space efficient algorithm to compute the alignment score as **the cost-only dynamic programming algorithm**.

Can we recover the alignment given $O(n+m)$ space?

Yes. Idea: By recursion!

- Algorithm $\text{Align}(S, T)$
 - Based on the cost-only algorithm, find the mid-point j such that $\text{score}(S, T) = \text{score}(S[1..n/2], T[1..j]) + \text{score}(S[n/2+1..n], T[j+1..m])$.
 - $A = \text{Align}(S[1..n/2], T[1..j])$ and $B = \text{Align}(S[n/2+1..n], T[j+1..m])$.
 - Report $A \bullet B$



Mid-point

- Mid-point is the position **j** that partitions an alignment in two halves.

$$\begin{array}{ccc}
 S[1..\frac{n}{2}] & S[\frac{n}{2} + 1..n] & \\
 \text{xxx...xx} & \text{yyy...yy} & \\
 | & | & \\
 \text{xxx...xx} & \text{yyy...yy} & \\
 T[1..j] & T[j + 1..m] &
 \end{array}
 =
 \begin{array}{ccc}
 S[1..\frac{n}{2}] & & S[\frac{n}{2} + 1..n] \\
 \text{xxx...xx} & & \text{yyy...yy} \\
 | & & | \\
 \text{xxx...xx} & & \text{yyy...yy} \\
 T[1..j] & & T[j + 1..m]
 \end{array}
 +
 \begin{array}{ccc}
 & & \\
 & & \\
 & & \\
 & & \\
 & &
 \end{array}$$

- We have the following facts:

1. The alignment score of the left half is $\text{Score}\left(S\left[1..\frac{n}{2}\right], T[1..j]\right)$
2. The alignment score of the right half is $\text{Score}\left(S\left[\frac{n}{2} + 1..n\right], T[j + 1..m]\right)$
3. $\text{Score}(S[1..n], T[1..m]) = \max_{0 \leq j \leq m} \left\{ \text{Score}\left(S\left[1..\frac{n}{2}\right], T[1..j]\right) + \text{Score}\left(S\left[\frac{n}{2} + 1..n\right], T[j + 1..m]\right) \right\}$

How to find the mid-point

Note:

- $Score(S[1..n], T[1..m]) = \max_{0 \leq j \leq m} \left\{ Score\left(S\left[1..\frac{n}{2}\right], T[1..j]\right) + Score\left(S\left[\frac{n}{2} + 1..n\right], T[j + 1..m]\right) \right\}$

1. Do cost-only dynamic programming for the first half.
 - Then, we find $Score(S[1..n/2], T[1..j])$ for all j
2. Do cost-only dynamic programming for the reverse of the second half.
 - Then, we find $Score(S[n/2+1..n], T[j+1..m])$ for all j
3. Determine j which maximizes
 - $Score\left(S\left[1..\frac{n}{2}\right], T[1..j]\right) + Score\left(S\left[\frac{n}{2} + 1..n\right], T[j + 1..m]\right)$

Example (mid-point Step 1)

	_	A	G	C	A	T	G	C	_
_	0	-1	-2	-3	-4	-5	-6	-7	
A	-1	2	1	0	-1	-2	-3	-4	
C	-2	1	1	3	2	1	0	-1	
A	-3	0	0	2	5	4	3	2	
A	-4	-1	-1	1	4	4	3	2	
T									
C									
C									
_									

Example (mid-point Step 2)

	_	A	G	C	A	T	G	C	_
_									
A									
C									
A									
A	-4	-1	-1	1	4	4	3	2	
T		-1	0	1	2	3	0	0	-3
C		-2	-1	1	-1	0	1	1	-2
C		-4	-3	-2	-1	0	1	2	-1
_		-7	-6	-5	-4	-3	-2	-1	0

Example (mid-point Step 3)

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

Example (Recursively solve the two subproblems)

	_	A	G	C	A	T	G	C	_
_									
A									
C									
A									
A									
T									
C									
C									
_									

The alignment
of left half

A _ CAA
A G C A _

A _ CAA T C C
A G C A _ T G C
Final alignment

T C C

T G C

The alignment
of right half

Time Analysis

- Time for finding mid-point:
 - Step 1 takes $O(n/2 \cdot m)$ time
 - Step 2 takes $O(n/2 \cdot m)$ time
 - Step 3 takes $O(m)$ time.
 - In total, $O(nm)$ time.
- Let $\text{Time}(n, m)$ be the time needed to recover the alignment.
- $\text{Time}(n, m)$
= time for finding mid-point + time for solving the two subproblems
= $O(nm) + \text{Time}(n/2, j) + \text{Time}(n/2, m-j)$
- Thus, time complexity = $T(n, m) = O(nm)$

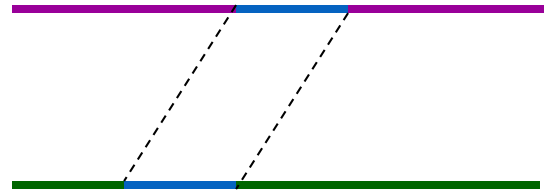
Space analysis

- Working memory for finding mid-point takes $O(m)$ space
- Once we find the mid-point, we can free the working memory
- Thus, in each recursive call, we only need to store the alignment path
- Observe that the alignment subpaths are disjoint, the total space required is $O(n+m)$.

More for string alignment problem

- Two special cases:
 - Longest common subsequence (LCS)
 - Score for mismatch is negative infinity
 - Score for insert/delete=0, Score for match=1
 - Hamming distance
 - Score for insert/delete is negative infinity
 - Score for match=1, Score for mismatch=0

Local alignment



- **Local alignment problem:** Find the best alignment between a substring of S and a substring of T.
- Application: Consider two proteins sharing a common domain.
 - Can you find this common domain?

Brute-force solution

- Algorithm:

- For every substring $A=S[i'..i]$ of S ,

- For every substring $B=T[j'..j]$ of T ,

- Compute the global alignment of A and B

- Return the pair (A, B) with the highest score

- Time:

- There are $n^2/2$ choices of A and $m^2/2$ choices of B .

- In total, there are $O(n^2m^2)$ substring pairs (A, B) .

- The global alignment of A and B can be computed in $O(nm)$ time.

- In total, time complexity = $O(n^3m^3)$

- Can we do better?

Some background

- X is a **suffix** of $S[1..n]$ if $X=S[k..n]$ for some $k \geq 1$
- X is a **prefix** of $S[1..n]$ if $X=S[1..k]$ for some $k \leq n$
- E.g.
 - Consider $S[1..7] = \text{ACCGATT}$
 - ACC is a prefix of S, GATT is a suffix of S
 - Empty string ε is both prefix and suffix of S

Recall Global alignment

- In global alignment, $V(i,j)$ is the highest alignment score between $S[1..i]$ and $T[1..j]$.

- Example 1: Consider $V(4,4)$.
 $V(4,4) = 4 - 3 = 1$

TGC_A

A_CAA

—	0
A	1
C	2
A	3
A	4
T	5
C	6
G	7

—	T	G	C	A	T	G	C
0	1	2	3	4	5	6	7

$V(4,4)$ is the highest alignment score between $S[1..4]$ and $T[1..4]$

Recall Global alignment

- In global alignment, $V(i,j)$ is the highest alignment score between $S[1..i]$ and $T[1..j]$.

- Example 2: Consider $V(4,2)$.
 $V(4,2) = -4$

ACAA

T__G

—	0
A	1
C	2
A	3
A	4
T	5
C	6
G	7

		T	G	C	A	T	G	C
	0	1	2	3	4	5	6	7

-4

$V(4,2)$ is the highest alignment score between $S[1..4]$ and $T[1..2]$

Change in local alignment

- In local alignment, $V(i,j)$ is modified. $V(i, j)$ is the highest alignment score between some suffix of $S[1..i]$ and some suffix of $T[1..j]$.

- Example 1: Consider $V(4,4)$.
- The suffix pair with the highest alignment score is $(S[2..4], T[3..4])$.

CAA

C _ A

- $V(3,4)=3$

—	0
A	1
C	2
A	3
A	4
T	5
C	6
G	7

		T G C A				T	G	C
—	0	1	2	3	4	5	6	7

3

$V(4,4)$ is the highest alignment score between some suffix of $S[1..4]$ and some suffix of $T[1..4]$

Change in local alignment

- In local alignment, $V(i,j)$ is modified. $V(i, j)$ is the highest alignment score between some suffix of $S[1..i]$ and some suffix of $T[1..j]$.
- Example 2: Consider $V(4,2)$.
- Alignment score between any non-empty suffix of $S[1..4]$ and any non-empty suffix of $T[1..2]$ is negative.
- We set $V(4,2)=0$.

		<div> <div></div> <div>T</div> <div>G</div> <div>C</div> <div>A</div> <div>T</div> <div>G</div> <div>C</div> </div>							
		0	1	2	3	4	5	6	7
—	0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	2	1	0	0
C	2	0	0	0	2	1	1	0	2
A	3	0	0	0	1	4	3	2	1
A	4	0	0	0	0	3	3	2	1
T	5	0	2	1	0	2	5	4	3
C	6	0	1	1	3	2	4	4	6
G	7	0	0	2	2	2	3	6	5

$V(4,2)$ is the highest alignment score between some suffix of $S[1..4]$ and some suffix of $T[1..2]$

Local alignment score

- $\max\{ V(i,j) \mid 0 \leq i \leq n, 0 \leq j \leq m \}$ is the highest alignment score among all substrings of S and all substrings of T.
- The local alignment score is the highest score in the entire table.
- For this example, the local alignment score is 6.

		—	T	G	C	A	T	G	C
		0	1	2	3	4	5	6	7
—	0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	2	1	0	0
C	2	0	0	0	2	1	1	0	2
A	3	0	0	0	1	4	3	2	1
A	4	0	0	0	0	3	3	2	1
T	5	0	2	1	0	2	5	4	3
C	6	0	1	1	3	2	4	4	6
G	7	0	0	2	2	2	3	6	5

How to fill in the V table

- Recall that $V(i,j)$ is the highest alignment score among all suffixes of $S[1..i]$ and all suffixes of $T[1..j]$.
- To compute $V(i,j)$ naively, we need to obtain the alignment scores between all suffixes of $S[1..i]$ and all suffixes of $T[1..j]$; then find their maximum.
- This is time consuming.

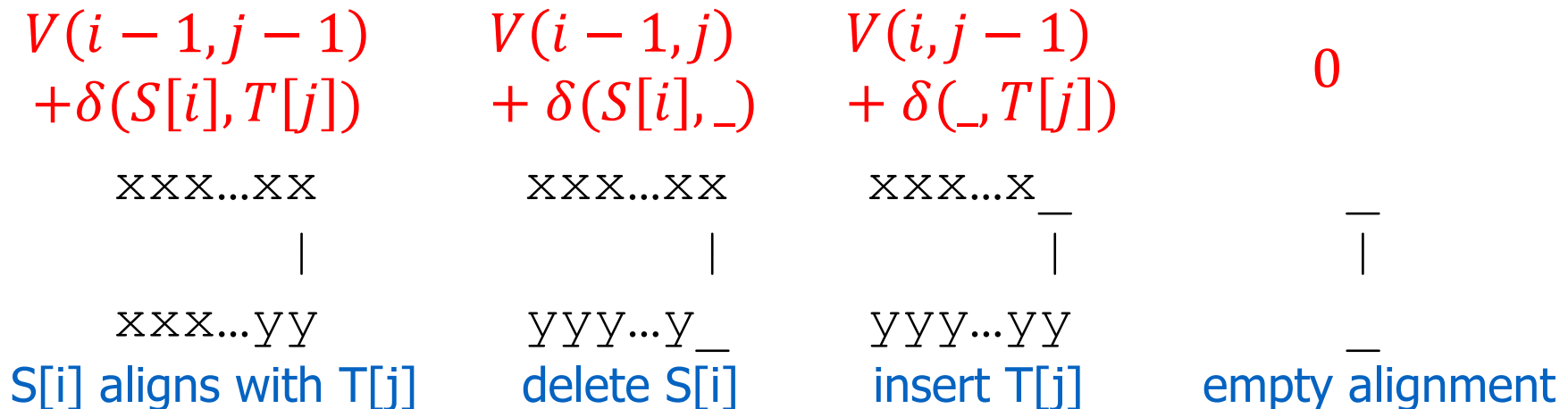
		_	T	G	C	A	T	G	C
		0	1	2	3	4	5	6	7
—	0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	2	1	0	0
C	2	0	0	0	2	1	1	0	2
A	3	0	0	0	1	4	3	2	1
A	4	0	0	0	0	3	3	2	1
T	5	0	2	1	0	2	5	4	3
C	6	0	1	1	3	2	4	4	6
G	7	0	0	2	2	2	3	6	5

Base case for $V(i,j)$

- Base case ($i=0$ or $j=0$):
 - $V(0,0)=0$ when $i=0$ and $j=0$
 - $V(i, 0) = 0$ when $j=0$
 - $V(0, j) = 0$ when $i=0$

Recursive case for $V(i,j)$

- Recursive case ($i > 0$ and $j > 0$):
- There are four cases:
 - (1) $S[i]$ aligns with $T[j]$,
 - (2) deletion of $S[i]$
 - (3) insertion of $T[j]$ or
 - (4) no alignment



Recursive case for $i > 0$ and $j > 0$

- Recursion for $i > 0$ and $j > 0$:

$$V(i, j) = \max \begin{cases} 0 & \text{Align empty strings} \\ V(i-1, j-1) + \delta(S[i], T[j]) & \text{Align } S[i] \text{ with } T[j] \\ V(i-1, j) + \delta(S[i], _) & \text{Delete } S[i] \\ V(i, j-1) + \delta(_, T[j]) & \text{Insert } T[j] \end{cases}$$

Smith-Waterman algorithm

1. $V(0,0)=0$

2. For $i = 1$ to n

- $V(i,0)=0$

3. For $j = 1$ to m

- $V(0, j)=0$

4. For $i = 1$ to n

- For $j = 1$ to m

- $V(i, j) = \max \begin{cases} 0 \\ V(i-1, j-1) + \delta(S[i], T[j]) \\ V(i-1, j) + \delta(S[i], _) \\ V(i, j-1) + \delta(_, T[j]) \end{cases}$

Base case

Recursive case

Example (I)

		—	T	G	C	A	T	G	C
		0	1	2	3	4	5	6	7
—	0	0	0	0	0	0	0	0	0
A	1	0							
C	2	0							
A	3	0							
A	4	0							
T	5	0							
C	6	0							
G	7	0							

	—	A	C	G	T
—		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

- Score for match = 2
- Score for insert, delete, mismatch = -1

Example (II)

$$\begin{aligned}
 &V(5,2) \\
 &= \max \begin{cases} 0 \\ V(4,1) + \delta(T,T) \\ V(4,2) + \delta(T,-) \\ V(5,1) + \delta(-,T) \end{cases} \\
 &= 1
 \end{aligned}$$

$$\begin{aligned}
 &V(5,3) \\
 &= \max \begin{cases} 0 \\ V(4,2) + \delta(T,C) \\ V(4,3) + \delta(T,-) \\ V(5,2) + \delta(-,C) \end{cases} \\
 &= 0
 \end{aligned}$$

$$\begin{aligned}
 &V(5,4) \\
 &= \max \begin{cases} 0 \\ V(4,3) + \delta(T,A) \\ V(4,4) + \delta(T,-) \\ V(5,3) + \delta(-,A) \end{cases} \\
 &= 2
 \end{aligned}$$

		—	T	G	C	A	T	G	C
		0	1	2	3	4	5	6	7
—	0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	2	1	0	0
C	2	0	0	0	2	1	1	0	2
A	3	0	0	0	1	4	3	2	1
A	4	0	0	0	0	3	3	2	1
T	5	0	2	1	0	2			
C	6								
G	7								

	—	A	C	G	T
—		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

- Score for match = 2
- Score for insert, delete, mismatch = -1

Example (III)

		_	T	G	C	A	T	G	C
		0	1	2	3	4	5	6	7
_	0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	2	1	0	0
C	2	0	0	0	2	1	1	0	2
A	3	0	0	0	1	4	3	2	1
A	4	0	0	0	0	3	3	2	1
T	5	0	2	1	0	2	5	4	3
C	6	0	1	1	3	2	4	4	6
G	7	0	0	2	2	2	3	6	5

	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

- Score for match = 2
- Score for insert, delete, mismatch = -1

Analysis

- We need to fill in all entries in the table with $n \times m$ matrix.
- Each entries can be computed in $O(1)$ time.
- Finally, finding the entry with the maximum value.
- Time complexity = $O(nm)$
- Space complexity = $O(nm)$

Recover the alignment

Given the V table, we can recover the alignment by calling $\text{back_tracking}(V, i, j)$ where (i, j) is the pair that maximizes $V(i, j)$.

- $\text{back_tracking}(V, i, j)$
 - $A = []$
 - While ($V(i, j) \neq 0$)
 - If ($V(i, j) = V(i, j-1) + \delta(_, T[j])$) /* Case 3: insert $T[j]$ */
 - $A = (_, T[j]) \bullet A$
 - $j = j - 1$
 - else if ($V(i, j) = V(i-1, j) + \delta(S[i], _)$) /* Case 2: delete $S[i]$ */
 - $A = (S[i], _) \bullet A$
 - $i = i - 1$
 - else if ($V(i, j) = V(i-1, j-1) + \delta(S[i], T[j])$) /* Case 1: $S[i]$ aligns with $T[j]$ */
 - $A = (S[i], T[j]) \bullet A$
 - $i = i - 1; j = j - 1;$
 - Return A

Example of back_tracking(V, 7, 6)

$V(7,6)$
 ↓ Case 1: report (G,G)
 $V(6,5)$
 ↓ Case 2: report (C,_)
 $V(5,5)$ — 0
 ↓ Case 1: report (T,T) A 1
 $V(4,4)$ C 2
 ↓ Case 1: report (A,A)
 $V(3,3)$ A 3
 ↓ Case 2: report (A,_)
 $V(2,3)$ A 4
 ↓ Case 1: report (C,C)
 $V(1,2)$ T 5
 C 6
 G 7

		—	T	G	C	A	T	G	C
		0	1	2	3	4	5	6	7
—	0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	2	1	0	0
C	2	0	0	0	2	1	1	0	2
A	3	0	0	0	1	4	3	2	1
A	4	0	0	0	0	3	3	2	1
T	5	0	2	1	0	2	5	4	3
C	6	0	1	1	3	2	4	4	6
G	7	0	0	2	2	2	3	6	5

C A A T C G
 C _ A T _ G

Semi-global alignment

δ	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

- We score alignments ignoring some end spaces
- **Example 1:** ignoring beginning and ending spaces on the second sequence.
 - ATCCGAA_ CATCAATCGAAGC
_____A[—]G[—]CAT[—]G[—]CAAT_____
 - If score the alignment normally, the score of the above alignment is 2
 - 8 matches (score=16), 12 insert (score=-12), 1 delete (score=-1), 1 mismatch (score=-1)
 - If we ignore the beginning and ending spaces on the second sequence, the score of the above alignment is 14
 - 8 matches (score=16), 1 delete (score=-1), 1 mismatch (score=-1)
 - This type of alignment can be used to locate gene in a prokaryotic genome

Semi-global alignment

δ	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

- Example 2: ignoring beginning spaces of the 1st sequence and ending spaces of the 2nd sequence
 - ACCTCACGATCCGA
TCAACGATCACCGCA
 - If score the alignment normally, the score of the above alignment is -8
 - 5 matches (score=10), 1 mismatch (score=-1), 9 insert (score=-9), 8 delete (score=-8)
 - If we ignoring beginning spaces on the 1st sequence and ending spaces on the 2nd sequence, the score of the above alignment is 9
 - 5 matches (score=10), 1 mismatch (score=-1)
 - This type of alignment can be used to find the common region of two overlapping sequences

How to compute semi-global alignment ignoring ending spaces?

- We just perform Needleman-Wunsch algorithm.
- To ignore ending spaces in S (first sequence), look for the maximum in the last row.
- Example: S=CAGC and T=GCAA
- The maximum in the last row is 2.
- The alignment is

CAGC--
--GCAA

- To ignore ending spaces in T (second sequence), look for the maximum in the last column.
- The maximum in the last column is 2.
- The alignment is

-C-AGC
GCAA--

		_	G	C	A	A
		0	1	2	3	4
_	0	0	-1	-2	-3	-4
C	1	-1	-1	1	0	-1
A	2	-2	-2	0	3	2
G	3	-3	0	-1	2	2
C	4	-4	-1	2	1	1

How to compute semi-global alignment ignoring beginning spaces?

- To ignore beginning spaces in S (first sequence), we perform Needleman-Wunsch algorithm, but initialize the first row to 0.
- Example: S=CAGC and T=GCAA
- $V(4,4)=2$.
- The alignment is:

–CAGC

GCA–A

		–	G	C	A	A
		0	1	2	3	4
–	0	0	0	0	0	0
C	1	-1	-1	2	1	0
A	2	-2	-2	1	4	3
G	3	-3	0	0	3	3
C	4	-4	-1	2	2	2

How to compute semi-global alignment ignoring beginning spaces?

- To ignore beginning spaces in T (second sequence), we perform Needleman-Wunsch algorithm, but we set the first column to 0.
- Example: S=CAGC and T=GCAA
- $V(4,4)=2$.
- The alignment is:

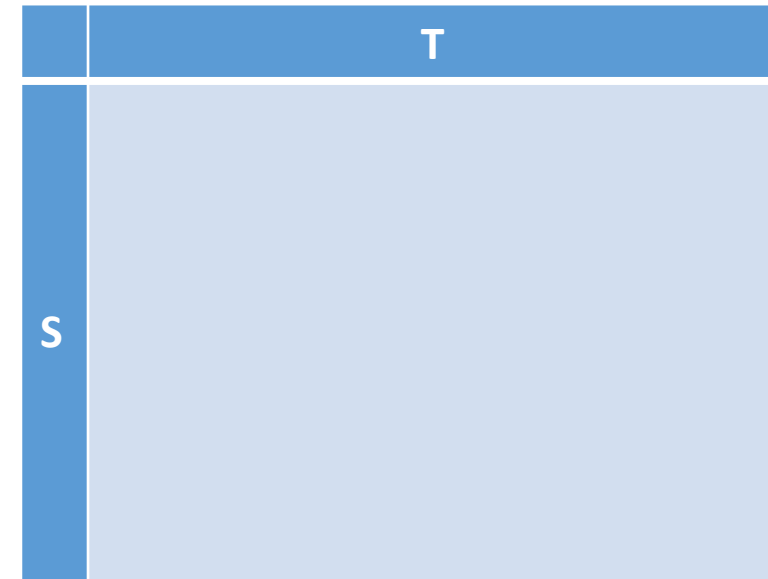
CAGC--

--GCAA

		—	G	C	A	A
		0	1	2	3	4
—	0	0	-1	-2	-3	-4
C	1	0	-1	1	0	-1
A	2	0	-1	0	3	2
G	3	0	2	1	2	2
C	4	0	1	4	3	2

Summary for semi-global alignment

- Below table summaries the changes to Needleman-Wunsch algorithm when we compute semi-global alignment.
- Note that you can use any combination of these 4 rules to compute different types of semi-global alignment.
- For example, to compute semi-global alignment ignoring all types of end spaces, we apply all 4 rules.



Spaces that are not charged	Modification to Needleman-Wunsch algorithm
Spaces in the beginning of $S[1..n]$	Initialize first row with zeros
Spaces in the ending of $S[1..n]$	Look for maximum in the last row
Spaces in the beginning of $T[1..m]$	Initialize first column with zeros
Spaces in the ending of $T[1..m]$	Look for maximum in the last column

Gaps

- A **gap** in an alignment is a maximal substring of contiguous spaces in either sequence of the alignment

This is a gap!



A sequence alignment diagram showing two sequences. The top sequence is 'A _CAACTCGCCTCC' and the bottom sequence is 'AGCA_____TGC'. The alignment is indicated by a blue line connecting the 'A' in the top sequence to the 'A' in the bottom sequence, and the 'TCC' in the top sequence to the 'TGC' in the bottom sequence. A purple arrow points from the text 'This is a gap!' to the space character between 'A' and 'C' in the top sequence. Another purple arrow points from the text 'This is another gap!' to the space character between 'CA' and 'T' in the bottom sequence.

```
A _CAACTCGCCTCC
AGCA_____TGC
```

This is another gap!

Penalty for gaps

- Previous discussion assumes the penalty for insert/delete is proportional to the length of a gap!
- This assumption may not be valid in some applications, for examples:
 - Mutation may cause insertion/deletion of a long substring. Such kind of mutation may be as likely as insertion/deletion of a single base.
 - Recall that mRNA splices out the introns. When aligning mRNA with its gene, the penalty should not be proportional to the length of the gaps.

Example

δ	A	C	G	T
A	2	-1	-1	-1
C	-1	2	-1	-1
G	-1	-1	2	-1
T	-1	-1	-1	2

- Definition: $g(q)$ is denoted as the penalty of a gap of length q
- Consider the following alignment.

```

      A _CAACTCGCC TCC
      A GCA_____TGC
  
```

- This alignment has 4 matches, 1 mismatch, 1 size-1 gap and 1 size-7 gap.
- If $g(q)=q$ (the gap penalty is proportion to the length of the gap),
 - Alignment score = $4*2-1-1-7=-1$
- If $g(q)=2$ (every gap has the same penalty),
 - Alignment score = $4*2-1-2-2=3$
- If $g(q)=0.5+0.5*q$ (an example gap penalty function that is not proportional to its length),
 - Alignment score = $4*2-1-1-4=2$

Optimal global alignment with general gap penalty

- With gap penalty $g(q)$, the global alignment of $S[1..n]$ and $T[1..m]$ can be found by dynamic programming.
- Denote $V(i, j)$ be the score for global alignment between $S[1..i]$ and $T[1..j]$.
- Base cases ($i=0$ or $j=0$):
 - $V(0, 0) = 0$
 - $V(i, 0) = -g(i)$
 - $V(0, j) = -g(j)$

Optimal global alignment with general gap penalty

- Recurrence: for $i > 0$ and $j > 0$,

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{S[i] aligns with T[j]} \\ \max_{0 \leq k \leq j-1} \{V(i, k) - g(j-k)\} & \text{Insert T[k+1..j]} \\ \max_{0 \leq k \leq i-1} \{V(k, j) - g(i-k)\} & \text{Delete S[k+1..i]} \end{cases}$$

DP algorithm for aligning S and T with general gap function $g(q)$

1. $V(0,0)=0$
2. For $i= 1$ to n
 - $V(i,0)= - g(i)$
3. For $j = 1$ to m
 - $V(0, j) = - g(j)$
4. For $i = 1$ to n
 - For $j = 1$ to m

$$\bullet V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) \\ \max_{0 \leq k \leq j-1} \{V(i, k) - g(j-k)\} \\ \max_{0 \leq k \leq i-1} \{V(k, j) - g(i-k)\} \end{cases}$$

5. Report $V(n,m)$

Base case

Recursive case

Analysis

- We need to fill in all entries in the $n \times m$ table.
- Space complexity = $O(nm)$
- Each entry can be computed in $O(n+m)$ time.
- Time complexity = $O(nm(n+m))$

Example (base case)

- $S = \text{ACCGA}$
- $T = \text{AGTTA}$
- Match=1, Mismatch=-3,
- $g(q) = 1 + q$
- Fill-in the 0th row and the 0th column.

		_	A	G	T	T	A
		0	1	2	3	4	5
_	0	0	-2	-3	-4	-5	-6
A	1	-2					
C	2	-3					
C	3	-4					
G	4	-5					
A	5	-6					

V table

Example (recursive case)

- S=ACCGA, T=AGTTA
- Match=1, Mismatch=-3,
- $g(q)=1+q$
- Fill-in row by row.

- Example:

$$V(3,1) = \max \begin{cases} V(2,0) + \delta(C,A) \\ \max_{0 \leq k \leq 1-1} \{V(3,k) - g(1-k)\} \\ \max_{0 \leq k \leq 3-1} \{V(k,1) - g(3-k)\} \end{cases}$$

$$V(3,1) = \max \begin{cases} V(2,0) + \delta(C,A) \\ V(3,0) - g(1) \\ V(0,1) - g(3), V(1,1) - g(2), V(2,1) - g(1) \end{cases} = V(1,1) - g(2) = -2$$

		_	A	G	T	T	A
		0	1	2	3	4	5
_	0	0	-2	-3	-4	-5	-6
A	1	-2	1	-1	-2	-3	-4
C	2	-3	-1	-2	-4	-5	-6
C	3	-4	?				
G	4	-5					
A	5	-6					

V table

Example (recursive case)

- $S=ACCGA$, $T=AGTTA$
- Match=1, Mismatch=-3,
- $g(q)=1+q$
- Fill-in row by row.

		_	A	G	T	T	A
		0	1	2	3	4	5
_	0	0	-2	-3	-4	-5	-6
A	1	-2	1	-1	-2	-3	-4
C	2	-3	-1	-2	-4	-5	-6
C	3	-4	-2	-4	-5	-6	-7
G	4	-5	-3	-1	-3	-4	-5
A	5	-6	-4	-3	-4	-6	-3

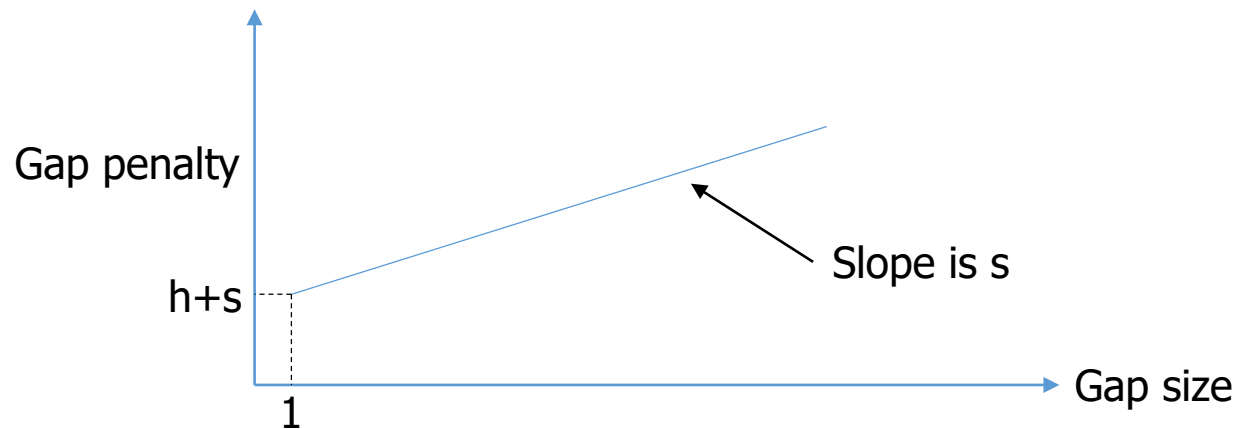
V table

ACCG--A
A--GTTA

Affine gap model

- In this model, the penalty for a gap is divided into two parts:
 - A penalty (h) for initiating the gap
 - A penalty (s) depending on the length of the gap
- Consider a gap with q spaces,
 - The penalty $g(q) = h + qs$

Usually, $h > s$



Dynamic programming solution (I)

- $V(i, j)$ is the score of a global optimal alignment between $S[1..i]$ and $T[1..j]$
- $E(i, j)$ is the score of a global optimal alignment between $S[1..i]$ and $T[1..j]$ with a space matches with $T[j]$
- $F(i, j)$ is the score of a global optimal alignment between $S[1..i]$ and $T[1..j]$ with $S[i]$ matches with a space

xxx...xx	xxx...x_
yyy...y_	yyy...yy
delete	insert
$F(i, j)$	$E(i, j)$

$V(i,j)$

- Basis:

- $V(0, 0) = 0$; $V(i, 0) = -h-is$; $V(0, j) = -h-js$

- Recurrence:

- $V(i, j) = \max \{ V(i-1, j-1) + \delta(S[i], T[j]), F(i, j), E(i, j) \}$

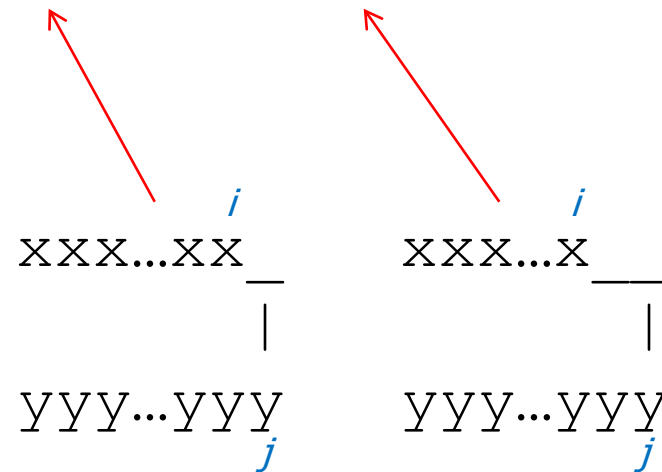
xxx...xx
|
xxx...yy
match/mismatch
 $V(i-1,j-1)+\delta(S[i], T[j])$

xxx...xx
|
yyy...y_
delete
 $F(i,j)$

xxx...x_
|
yyy...yy
insert
 $E(i,j)$

$E(i,j)$

- Basis:
 - $E(0, 0) = -\infty$; $E(i, 0) = -\infty$; $E(0, j) = -h-j$
- Recurrence ($i > 0$ and $j > 0$):
 - $E(i, j) = \max \{ V(i, j-1) - h - s, E(i, j-1) - s \}$



Case 1

Case 2

$F(i,j)$

- Basis:

- $F(0, 0) = -\infty$; $F(i, 0) = -h-is$; $F(0, j) = -\infty$

- Recurrence ($i>0, j>0$):

- $F(i, j) = \max \{ V(i-1, j)-h-s, F(i-1, j)-s \}$

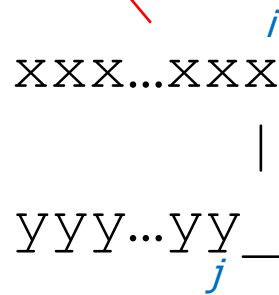


Diagram illustrating Case 1 of the recurrence. It shows a sequence of 'x's and 'y's. The top row consists of 'x's, with the last 'x' indexed by a blue i . A vertical line connects this 'x' to the last 'y' in the bottom row. The bottom row consists of 'y's, with the last 'y' indexed by a blue j . A horizontal line follows the last 'y'.

Case 1

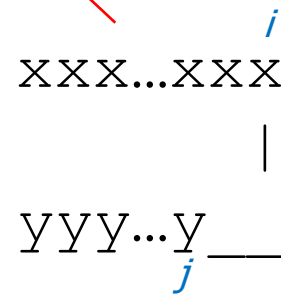


Diagram illustrating Case 2 of the recurrence. It shows a sequence of 'x's and 'y's. The top row consists of 'x's, with the last 'x' indexed by a blue i . A vertical line connects this 'x' to the last 'y' in the bottom row. The bottom row consists of 'y's, with the last 'y' indexed by a blue j . A horizontal line follows the last 'y'.

Case 2

Summary of the dynamic programming solution

- Basis:
 - $V(0, 0) = 0$; $V(i, 0) = -h-is$; $V(0, j) = -h-js$
 - $E(0, 0) = -\infty$; $E(i, 0) = -\infty$; $E(0, j) = -h-js$
 - $F(0, 0) = -\infty$; $F(i, 0) = -h-is$; $F(0, j) = -\infty$
- Recurrence:
 - $E(i, j) = \max \{ E(i, j-1)-s, D(i, j-1)-h-s \}$
 - $F(i, j) = \max \{ F(i-1, j)-s, D(i-1, j)-h-s \}$
 - $V(i, j) = \max \{ V(i-1, j-1) + \delta(S[i], T[j]), E(i, j), F(i, j) \}$

Global alignment algorithm with affine gap penalty

1. $V(0, 0) = 0$; $E(0, 0) = -\infty$; $F(0, 0) = -\infty$;

2. For $i = 1$ to n

- $V(i, 0) = -h-is$; $E(i, 0) = -\infty$; $F(i, 0) = -h-is$;

3. For $j = 1$ to m

- $V(0, j) = -h-js$; $E(0, j) = -h-js$; $F(0, j) = -\infty$;

4. For $i = 1$ to n

- For $j = 1$ to m

- $E(i, j) = \max \{ E(i, j-1) - s, V(i, j-1) - h - s \}$

- $F(i, j) = \max \{ F(i-1, j) - s, V(i-1, j) - h - s \}$

- $V(i, j) = \max \{ V(i-1, j-1) + \delta(S[i], T[j]), E(i, j), F(i, j) \}$

5. Report $V(n, m)$

Base case

Recursive case

Example

- S=ACCGA
- T=AGTTA
- Match=1, Mismatch=-3,
initial gap (h) =1, each
space (s) = 1
- Fill-in base case.

		_	A	G	T	T	A
		0	1	2	3	4	5
_	0	0	-2	-3	-4	-5	-6
A	1	-2					
C	2	-3					
C	3	-4					
G	4	-5					
A	5	-6					

V table

		_	A	G	T	T	A
		0	1	2	3	4	5
_	0	-∞	-2	-3	-4	-5	-6
A	1	-∞					
C	2	-∞					
C	3	-∞					
G	4	-∞					
A	5	-∞					

E table

$$V(0, 0) = 0; V(i, 0) = -h-is; V(0, j) = -h-js$$

$$E(0, 0) = -\infty; E(i, 0) = -\infty; E(0, j) = -h-js$$

$$F(0, 0) = -\infty; F(i, 0) = -h-is; F(0, j) = -\infty$$

		_	A	G	T	T	A
		0	1	2	3	4	5
_	0	-∞	-∞	-∞	-∞	-∞	-∞
A	1	-2					
C	2	-3					
C	3	-4					
G	4	-5					
A	5	-6					

F table

Example

- S=ACCGA
- T=AGTTA
- Match=1,
Mismatch=-3,
initial gap (h) =-1,
each space (s) = -1
- Fill-in row by row.
- For each entry,
 - fill-in E, F
 - Then, V

$$E(i, j) = \max \{ E(i, j-1) - s, V(i, j-1) - h - s \}$$

$$F(i, j) = \max \{ F(i-1, j) - s, V(i-1, j) - h - s \}$$

$$V(i, j) = \max \{ V(i-1, j-1) + \delta(S[i], T[j]), F(i, j), E(i, j) \}$$

		_	A	G	T	T	A
		0	1	2	3	4	5
_	0	0	-2	-3	-4	-5	-6
A	1	-2	1				
C	2	-3					
C	3	-4					
G	4	-5					
A	5	-6					

V table

		_	A	G	T	T	A
		0	1	2	3	4	5
_	0	-∞	-2	-3	-4	-5	-6
A	1	-∞	-4				
C	2	-∞					
C	3	-∞					
G	4	-∞					
A	5	-∞					

E table

		_	A	G	T	T	A
		0	1	2	3	4	5
_	0	-∞	-∞	-∞	-∞	-∞	-∞
A	1	-2	-4				
C	2	-3					
C	3	-4					
G	4	-5					
A	5	-6					

F table

$$E(i, j) = \max \{ E(i, j-1) - s, V(i, j-1) - h - s \}$$

$$F(i, j) = \max \{ F(i-1, j) - s, V(i-1, j) - h - s \}$$

$$V(i, j) = \max \{ V(i-1, j-1) + \delta(S[i], T[j]), F(i, j), E(i, j) \}$$

Example

- S=ACCGA
- T=AGTTA
- Match=1, Mismatch=-3, initial gap (h) =-1, each space (s) = -1
- Fill-in row by row.
- For each entry,
 - fill-in E, F
 - Then, V

		A	G	T	T	A
	0	1	2	3	4	5
0	0	-2	-3	-4	-5	-6
A 1	-2	1	-1	-2	-3	-4
C 2	-3	-1	-2	-4	-5	-6
C 3	-4	-2	-4	-5	-6	-7
G 4	-5	-3	-1	-3	-4	-5
A 5	-6	-4	-3	-4	-6	-3

V table

		A	G	T	T	A
	0	1	2	3	4	5
0	-∞	-2	-3	-4	-5	-6
A 1	-∞	-4	-1	-2	-3	-4
C 2	-∞	-5	-3	-4	-5	-6
C 3	-∞	-6	-4	-5	-6	-7
G 4	-∞	-7	-5	-3	-4	-5
A 5	-∞	-8	-6	-5	-6	-7

E table

		A	G	T	T	A
	0	1	2	3	4	5
0	-∞	-∞	-∞	-∞	-∞	-∞
A 1	-2	-4	-5	-6	-7	-8
C 2	-3	-1	-3	-4	-5	-6
C 3	-4	-2	-4	-5	-6	-7
G 4	-5	-3	-5	-6	-7	-8
A 5	-6	-4	-3	-5	-6	-7

F table

ACCG--A
A--GTTA

Analysis

- We need to fill in 3 tables, each is of size $n \times m$.
- Space complexity = $O(nm)$
- Each entry can be computed in $O(1)$ time.
- Time complexity = $O(nm)$

Scoring function

- In the rest of this lecture, we discuss the scoring function for both DNA and Protein

Scoring function for Protein

- Scoring function for amino acid is a 20x20 matrix.
- There are a few scoring matrix:
 - Identity matrix
 - Genetic code matrix
 - Physical property matrix: based on chemical/physical similarity
 - Statistical scoring matrices (PAM, BLOSUM): based on observed substitution frequencies

Identity matrix

- The simple amino-acid scoring matrix is the identity matrix
 - $\delta(x,y)=1$ if $x=y$; otherwise, $\delta(x,y)=0$.
- This scoring scheme cannot capture that fact that some amino acids are closely related. For example,
 - N and D
 - E and Q

[illegible]

Genetic code matrix

- Score is defined by the number of shared nucleotides in their codons.
- E.g. $\delta(F,S)=2$
 - The codon of F is TTT
 - The codon of S is TCT

	T	C	A	G	
T	TTT Phe [F]	TCT Ser [S]	TAT Tyr [Y]	TGT Cys [C]	T
	TTC Phe [F]	TCC Ser [S]	TAC Tyr [Y]	TGC Cys [C]	C
	TTA Leu [L]	TCA Ser [S]	TAA Ter [end]	TGA Ter [end]	A
	TTG Leu [L]	TCG Ser [S]	TAG Ter [end]	TGG Trp [W]	G
C	CTT Leu [L]	CCT Pro [P]	CAT His [H]	CGT Arg [R]	T
	CTC Leu [L]	CCC Pro [P]	CAC His [H]	CGC Arg [R]	C
	CTA Leu [L]	CCA Pro [P]	CAA Gln [Q]	CGA Arg [R]	A
	CTG Leu [L]	CCG Pro [P]	CAG Gln [Q]	CGG Arg [R]	G
A	ATT Ile [I]	ACT Thr [T]	AAT Asn [N]	AGT Ser [S]	T
	ATC Ile [I]	ACC Thr [T]	AAC Asn [N]	AGC Ser [S]	C
	ATA Ile [I]	ACA Thr [T]	AAA Lys [K]	AGA Arg [R]	A
	ATG Met [M]	ACG Thr [T]	AAG Lys [K]	AGG Arg [R]	G
G	GTT Val [V]	GCT Ala [A]	GAT Asp [D]	GGT Gly [G]	T
	GTC Val [V]	GCC Ala [A]	GAC Asp [D]	GGC Gly [G]	C
	GTA Val [V]	GCA Ala [A]	GAA Glu [E]	GGA Gly [G]	A
	GTG Val [V]	GCG Ala [A]	GAG Glu [E]	GGG Gly [G]	G

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	3	1	2	2	1	2	1	1	1	1	1	1	2	1	1	2	2	2	1	1
C	1	3	1	0	2	2	1	1	0	1	0	1	1	0	2	2	1	1	2	2
D	2	1	3	2	1	2	2	1	1	1	0	2	1	1	1	1	1	2	0	2
E	2	0	2	3	0	2	1	1	2	1	1	1	1	2	1	1	1	2	1	1
F	1	2	1	0	3	1	1	2	0	2	1	1	1	0	1	2	1	2	1	2
G	2	2	2	2	1	3	1	1	1	1	1	1	1	1	2	2	1	2	2	1
H	1	1	2	1	1	1	3	1	1	2	0	2	2	2	2	1	1	1	0	2
I	1	1	1	1	2	1	1	3	2	2	2	2	1	1	2	2	2	2	0	1
K	1	0	1	2	0	1	1	2	3	1	2	2	1	2	2	1	2	1	1	1
L	1	1	1	1	2	1	2	2	1	3	2	1	2	2	2	2	1	2	2	1
M	1	0	0	1	1	1	0	2	2	2	3	1	1	1	2	1	2	2	1	0
N	1	1	2	1	1	1	2	2	2	1	1	3	1	1	1	2	2	1	0	2
P	2	1	1	1	1	1	2	1	1	2	1	1	3	2	2	2	2	1	1	1
Q	1	0	1	2	0	1	2	1	2	2	1	1	2	3	2	1	1	1	1	1
R	1	2	1	1	1	2	2	2	2	2	2	1	2	2	3	2	2	1	2	1
S	2	2	1	1	2	2	1	2	1	2	1	2	2	1	2	3	2	1	2	2
T	2	1	1	1	1	1	1	2	2	1	2	2	2	1	2	2	3	1	1	1
V	2	1	2	2	2	2	1	2	1	2	2	1	1	1	1	1	1	3	1	1
W	1	2	0	1	1	2	0	0	1	2	1	0	1	1	2	2	1	1	3	1
Y	1	2	2	1	2	1	2	1	1	1	0	2	1	1	1	2	1	1	1	3

Scoring function for protein using physical/chemical properties

- Idea: an amino acid is more likely to be substituted by another if they have similar chemical/physical property
 - E.g. we give higher score for substituting nonpolar amino acid to another nonpolar amino acid
- George et al. (1990, Methods Enzymol. 183: 333)
 - The substitution score is assigned based on the relative hydrophobicity of the amino acids
- Karlin and Ghandour (1985, PNAS 82:8597)
 - The score function is derived based on hydrophobicity, charge, electronegativity, and size
- Doolittle (1990, Methods Enzymol. 183: 1)
 - The score is determined by the steric interactions of amino acid side chain structure and the ease of codon transformation.
- These models cannot give a good enough scoring function since it is too complex to model evolution and physical/chemical properties using a few parameters.

Scoring function for protein based on statistical model

- Most often used approaches
- Statistical methods define the similarity score as the log-odds ratio between the observed substitution rate $O(a,b)$ and the expected substitution rate $E(a,b)$: $\delta(a, b) = \log \frac{O(a,b)}{E(a,b)}$.
- As stated before, when such similarity score, the alignment algorithm can compute the alignment score correctly.

Scoring function for DNA

- For DNA, since we only have 4 nucleotides, the score function is a 4x4 matrix.
- Criteria for the scoring function for DNA:
 1. Positive for match
 2. Negative for mismatch
 3. The expected score is negative when an alignment is extended by random pairs.

Criteria 3

- We required the expected score for aligning a random pair of residues/bases to be negative.
- Otherwise, the longer the alignment, the higher is the score independent of whether the segments aligned are related or not.

	A	C	G	T
A	2	-1	-1	-1
C	-1	2	-1	-1
G	-1	-1	2	-1
T	-1	-1	-1	2

Good!

Expected score =

$$\frac{1}{16} (4 * 2 + 12 * (-1)) = -\frac{1}{4}$$

	A	C	G	T
A	5	-1	-1	-1
C	-1	5	-1	-1
G	-1	-1	5	-1
T	-1	-1	-1	5

Not Good!

Expected score =

$$\frac{1}{16} (4 * 5 + 12 * (-1)) = \frac{1}{2}$$

Score function for DNA

- M is a Markov model such that $M(x,y)$ is the probability that x is mutated to y .
- For DNA base x , p_x is the proportion of the base x .
- The log-odds score $S_{ij} = \log \left(\frac{p_i M(i,j)}{p_i p_j} \right)$.
- For DNA, assume the four nucleotides have the same frequency, we have $S_{ij} = \log(4M_{nij})$.

τ similarity

	A	C	G	T
A	τ	$(1 - \tau)/3$	$(1 - \tau)/3$	$(1 - \tau)/3$
C	$(1 - \tau)/3$	τ	$(1 - \tau)/3$	$(1 - \tau)/3$
G	$(1 - \tau)/3$	$(1 - \tau)/3$	τ	$(1 - \tau)/3$
T	$(1 - \tau)/3$	$(1 - \tau)/3$	$(1 - \tau)/3$	τ

- Match score: $S_{xx} = \log(4M(x, x)) = \log(4 * \tau)$. $M(x, y)$
- Mismatch score: $S_{xy} = \log(4M(x, y)) = \log\left(4 * \frac{(1-\tau)}{3}\right)$.
- Match/Mismatch ratio = $\frac{-\log(4*\tau)}{\log\left(4*\frac{(1-\tau)}{3}\right)}$.

99% similarity

	A	C	G	T
A	0.99	0.0033	0.0033	0.0033
C	0.0033	0.99	0.0033	0.0033
G	0.0033	0.0033	0.99	0.0033
T	0.0033	0.0033	0.0033	0.99

- So, $M(x,x)=0.99$
- Match/Mismatch ratio = $\frac{-\log(4*0.99)}{\log\left(4*\frac{(1-0.99)}{3}\right)} = 0.32.$
- Example: Match score is 1 and mismatch score is -3.

$M(x,y)$

	A	C	G	T
A	1	-3	-3	-3
C	-3	1	-3	-3
G	-3	-3	1	-3
T	-3	-3	-3	1

95% similarity for DNA

	A	C	G	T
A	0.95	0.0167	0.0167	0.0167
C	0.0167	0.95	0.0167	0.0167
G	0.0167	0.0167	0.95	0.0167
T	0.0167	0.0167	0.0167	0.95

- Match/Mismatch ratio = $\frac{-\log(4*0.95)}{\log\left(4*\frac{(1-0.95)}{3}\right)} = 0.493.$ $M(x,y)$
- Example: Match score is 1 and mismatch score is -2.
- MegaBLAST uses this score!

	A	C	G	T
A	1	-2	-2	-2
C	-2	1	-2	-2
G	-2	-2	1	-2
T	-2	-2	-2	1

89% similarity for DNA

	A	C	G	T
A	0.89	0.0367	0.0367	0.0367
C	0.0367	0.89	0.0367	0.0367
G	0.0367	0.0367	0.89	0.0367
T	0.0367	0.0367	0.0367	0.89

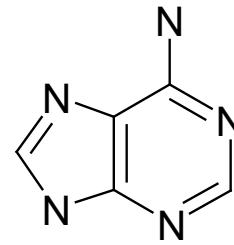
$M(x,y)$

- For 89% similarity.
- So, $M_{10}(x,x)=0.89$
- Match/Mismatch ratio = $\frac{-\log(4*0.89)}{\log\left(4*\frac{(1-0.89)}{3}\right)} = 0.66$.
- Example: Match score is 2 and mismatch score is -3.

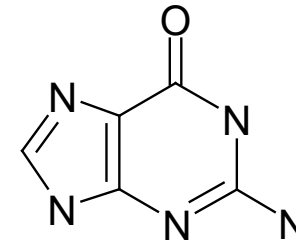
- BLAST uses this score!

	A	C	G	T
A	2	-3	-3	-3
C	-3	2	-3	-3
G	-3	-3	2	-3
T	-3	-3	-3	2

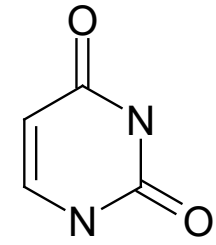
Scoring function for DNA



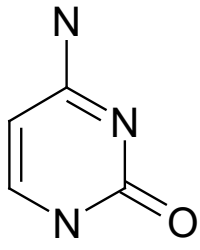
Adenine



Guanine



Thymine



Cytosine

- Since DNA has 4 nucleotides only, the score function is simple.
 - BLAST matrix
 - Transition Transversion matrix:
 - Transition = replacing purine (A,G) by purine or replacing pyrimidine (C,T) by pyrimidine!
 - Transversion = replacing purine by pyrimidine, or vice versa
 - Give mild penalty to transition

	A	C	G	T
A	2	-3	-3	-3
C	-3	2	-3	-3
G	-3	-3	2	-3
T	-3	-3	-3	2

BLAST score matrix

	A	C	G	T
A	1	-5	-1	-5
C	-5	1	-5	-1
G	-1	-5	1	-5
T	-5	-1	-5	1

Transition Transversion score matrix

PAM and BLOSUM

- Two popular matrices for protein:
 - Point Accepted Mutation (PAM) matrix
 - BLOSUM

Point Accepted Mutation (PAM)

- PAM was developed by Dayhoff (1978).
- A **point mutation** means the substitution of one residue by another.
- It is called an **accepted point mutation** if the mutation does not change the protein's function or is not fatal.
- A **PAM unit** is an evolutionary time period over which 1% of the sequence are expected to have accepted point mutations.

PAM matrix by example (I)

- Ungapped alignment is constructed for high similarity amino acid sequences (usually >85%)
- Below is a simplified global multiple alignment of some highly similar amino acid sequences (without gap):

- IACGCTAFK
IGCGCTAFK
LACGCTAFK
IGCGCTGFK
IGCGCTLFK
LASGCTAFK
LACACTAFK

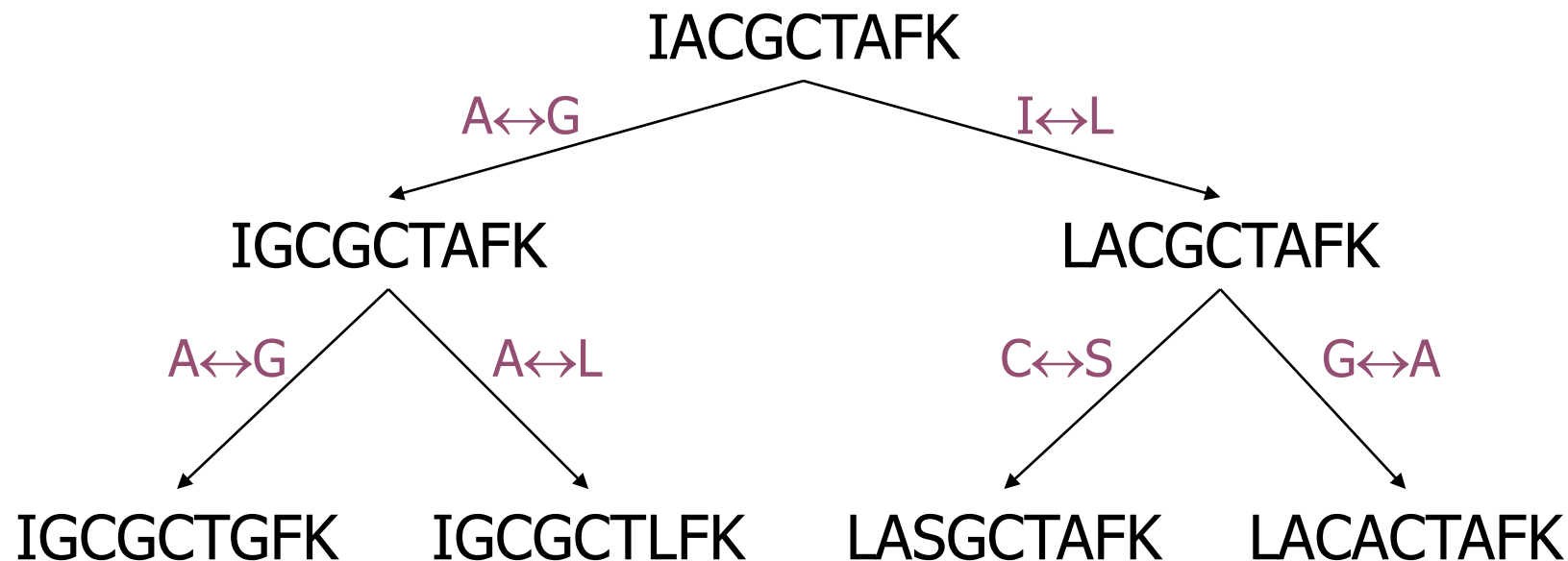
This alignment has 63 residues.

x	f_x
A	10/63
C	13/63
F	7/63
G	10/63
I	4/63
K	7/63
L	4/63
S	1/63
T	7/63

PAM matrix by example (II)

$$\sum_x \left(\sum_{y \neq x} F_{x,y} \right) = 12$$

- Build the phylogenetic tree for the sequences
- Then, denote $F_{x,y}$ be the observed number of mutations from x to y.



x	y	F_{xy}
A	G	3
A	L	1
C	S	1
G	A	3
I	L	1
L	A	1
L	I	1
S	C	1

PAM similarity matrices

- PAM is a family of similarity matrixes.
- If we allow n accepted point mutations per 100 residues, the corresponding substitution matrix is called PAM- n matrix.

PAM-1

- Let $O(a, b)$ be the observed substitution rate from the residue a to the residue b under 1 PAM divergence.
- Since PAM-1 assume 1 mutation per 100 residues, the probability of no substitution is 0.99, that is, $\sum_x O(x, x) = 0.99$.
- Also, the probability of having a substitution is 0.01, that is, $\sum_x (\sum_{y \neq x} O(x, y)) = 0.01$.
- For $a \neq b$, $O(a, b)$ should be proportional to $F_{a, b}$.
- Hence, we set $O(a, b) = \frac{0.01 \cdot F_{a, b}}{\sum_x (\sum_{y \neq x} F_{x, y})}$.

PAM-1

- Let $E(a, b)$ be the expected frequency of substituting residue a by residue b .
- $E(a, b) = f_a \cdot f_b$.
- The PAM-1 similarity score $\delta(a, b) = \log \frac{O(a, b)}{E(a, b)}$.

Example for PAM-1

- E.g., $F_{A,G} = 3$. $f_A = f_G = 10/63$.
- $O(A, G) = \frac{0.01 \cdot F_{A,G}}{\sum_x (\sum_{y \neq x} F_{x,y})} = \frac{0.01 \cdot F_{A,G}}{\sum_x (\sum_{y \neq x} F_{x,y})} = \frac{0.01 \cdot 3}{12} = 0.0025$
- $E(A, G) = f_A \cdot f_G = \frac{10}{63} \cdot \frac{10}{63} = 0.0252$
- $\delta(A, G) = \log \frac{O(A,G)}{E(A,G)} = \log \frac{0.0025}{0.0252} = \log(0.09925) = -1.0034$

PAM-2 matrix

- The PAM-2 matrix can be created by extrapolating the PAM-1 matrix.
- Let $M(a,b)$ be the probability that the residue a is mutated to the residue b in 1 PAM diverge. By definition, $M(a,b)$ equals $O(a,b) / f_a$.
- Suppose the residue a is mutated to another residue b in 2 PAM diverge. This means that the residue a is mutated to some residue x ; then the residue x is mutated to the residue b .
- Hence, $\text{Pr}(a \text{ is mutated to } b \text{ in 2 PAM diverge}) = \sum_x M(a, x) \cdot M(x, b) = M^2(a, b)$, where M^2 is the square of the matrix M by matrix multiplication.
- The observed substitution rate from a to b in 2 PAM diverge is $f_a M^2(a,b)$.
- The expected substitution rate from a to b is $f_a f_b$.
- The PAM-2 similarity score $\delta(a,b)$ is the log odds of the observed substitution rate and the expected substitution rate, which is
$$\log(f_a M^2(a,b) / f_a f_b) = \log(M^2(a,b) / f_b)$$

PAM-n matrix

- In general, the PAM-n matrix can be created by extrapolating the PAM-1 matrix.
- We can show that $\Pr(a \text{ is mutated to } b \text{ in } n \text{ PAM diverge}) = M^n(a,b)$.
- The observed substitution rate from a to b in n PAM diverge is $f_a M^n(a,b)$.
- The expected substitution rate from a to b is $f_a f_b$.
- The PAM-n similarity score $\delta(a,b)$ is the log odds of the observed substitution rate and the expected substitution rate, which is
$$\log(f_a M^n(a,b)/f_a f_b) = \log(M^n(a,b)/f_b)$$

%difference versus PAM

- For PAM-n matrix,
 - %similar = $\sum_{x \in A} M^n(x, x) f_x$
 - %difference = $1 - \sum_{x \in A} M^n(x, x) f_x$
- The right table shows the similarity and difference for PAM-n.
- Example:
 - When two sequences are 250 PAM units apart, the two sequences are approximately 20% similar or 80% difference.

PAM	%Difference	%similar
1	1	99
5	5	95
11	10	90
17	15	95
23	20	80
30	25	75
38	30	70
47	35	65
56	40	60
67	45	55
80	50	50
94	55	45
112	60	40
133	65	35
159	70	30
195	75	25
246	80	20
328	85	15

History of PAM

- Dayhoff, M. O., Schwartz, R. M., and Orcutt, B. C. [1979] in Atlas of Protein Sequence and Structure
 - This is the matrix MDM78 PAM250
- Gribskov, M. & Burgess, R.R. (1986) *NAR* **14**:6745-6763
 - They renormalize MDM78 so that $M(i,i)=1.5$
 - The scores for non-identical residues have been adjusted to give a mean of -0.17 and a standard deviation of 0.364. The negative expectation value is necessary for local alignment routines
- Jones, D.T., Taylor, W.R. & Thornton, J.M. (1992), The rapid generation of mutation data matrices from protein sequences. *CABIOS* **8**:275-282. They have automated the procedure of deriving scoring matrices from sequence databases.
 - Using 40 folds larger database, a better PAM is built.
- Gaston Gonnet and coworkers (Gonnet, G., Cohen, M. A.& Benner, S. (1992) *Science* **256**:1443-1445) have calculated an exhaustive all-against-all sequence matching of the entire protein database using their DARWIN system.
 - By using a large database, they can directly compile mutation matrix at different PAM distances.
 - They show that extrapolation introduces significant errors.

BLOSUM (BLOck SUBstitution Matrix)

- Since the PAM matrix is generated by extrapolation, PAM matrix did not work well for aligning sequences that are evolutionarily divergent.
- Henikoff and Henikoff (1992) proposed BLOSUM.
- Unlike PAM, BLOSUM matrix is constructed directly from the observed alignment (instead of extrapolation)

Generating conserved blocks

- In BLOSUM, the input is the set of multiple alignments for nonredundant groups of protein families.
- Based on PROTOMAT, blocks of nongapped local alignments are derived.
- Each block represents a conserved region of a protein family.

Extract frequencies from blocks

- From all blocks, we count the frequency p_a for each amino acid residue a .
- For any two amino acid residues a and b , we count the frequency p_{ab} of the aligned pairs of a and b .
- For example,
 - IACGCTAFK
IGCGCTAFK
LACGCTAFK
IGCGCTGFK
IGCGCTLFK
LASGCTAFK
LACACTAFK
- There are $7 \times 9 = 63$ residues, including 10's A and 10's G. Hence, $p_A = p_G = 10/63$.
- There are $9 \binom{7}{2} = 189$ aligned residue pairs, including 23 (A, G) pairs. Hence, $p_{AG} = 23/198$.

The scoring function of BLOSUM

- For each pair of aligned residues a and b , the alignment score $\delta(a,b) = 1/\lambda \ln p_{ab}/(p_a p_b)$, where
 - p_{ab} is the probability that a and b are observed to align together. p_a and p_b are the frequency of residues a and b respectively.
 - λ is a normalization constant. In BLOSUM65, λ is set to $\frac{\ln(2)}{2} = 0.347$. Then, $\delta(a,b)$ represents multiples of half bits.
- Example: $p_A=p_G=0.159$, $p_{AG} = 0.122$. With $\lambda=0.347$, $\delta(A,L)=4.538$.
 - This indicates that the A-G pairing in homologous sequences is about $2^{(4.538/2)}=4.82$ times more likely than by chance.

What is BLOSUM 62?

- To reduce multiple contributions to amino acid pair frequencies from the most closely related members of a family, similar sequences are merged within block.
- BLOSUM p matrix is created by merging sequences with no less than p% similarity.
- For example,
 - AVAAA
AVAAA
AVAAA
AVLAA
VVAAL
- Note that the first 4 sequences have at least 80% similarity. The similarity of the last sequence with the other 4 sequences is less than 62%.
- For BLOSUM 62, we group the first 4 sequences and we get
 - $AV[A_{0.75}L_{0.25}]AA$
VVAAL
- Then, $p_{AV} = 1 / 5$ and $P_{AL} = (0.25 + 1)/5$.

Relationship between BLOSUM and PAM

- Relationship between BLOSUM and PAM
 - BLOSUM 80 \approx PAM 120
 - BLOSUM 62 \approx PAM 160
 - BLOSUM 45 \approx PAM 250
- BLOSUM 62 is the default matrix for BLAST 2.0

BLOSUM 62

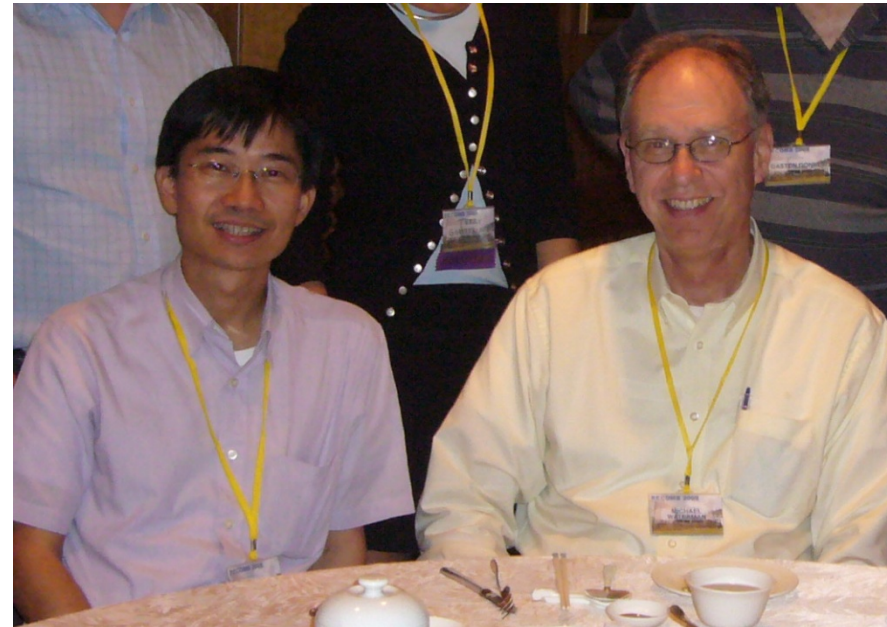
	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	4	0	-2	-1	0	-2	-1	-1	-1	-1	-2	-2	-1	-1	-1	1	0	0	-3	-2
C	0	9	-3	-4	-2	-3	-3	-1	-3	-1	-1	-3	-3	-3	-3	-1	-1	-1	-2	-2
D	-2	-3	6	2	-3	-1	-1	-3	-1	-4	-3	1	-1	0	-2	0	-1	-3	-4	-3
E	-1	-4	2	5	-3	-2	0	-3	1	-3	-2	0	-1	2	0	0	-1	-2	-3	-2
F	0	-2	-3	-3	6	-3	-1	0	-3	0	0	-3	-4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

Photos of Smith & Waterman

Limsoon & Temple Smith



Ken & Michael Waterman



References

- R. F. Doolittle, M. Hunkapiller, L. E. hood, S. Devare, K. Robbins, S. Aaronson, and H. Antoniades. Simian sarcoma virus onc gene v-sis, is derived from the gene (or genes) encoding a platelet-derived growth factor. *Science*, 221:275-277, 1983.
- S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443-453, 1970.
- T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195-197, 1981.

References

- M. Waterman, T. F. Smith, and W. A. Beyer. Some biological sequence metrics. *Advances in Mathematics*, 20:367-387, 1976.
- M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. In M. O. Dayhoff (ed) *Atlas of Protein Sequence and Structure*, volume 5, supplement 3, pp. 345-352. National Biomedical Research Foundation, Washington, DC, 1978.
- Henikoff, S. and Henikoff, J. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*. 89(biochemistry): 10915 - 10919 (1992).