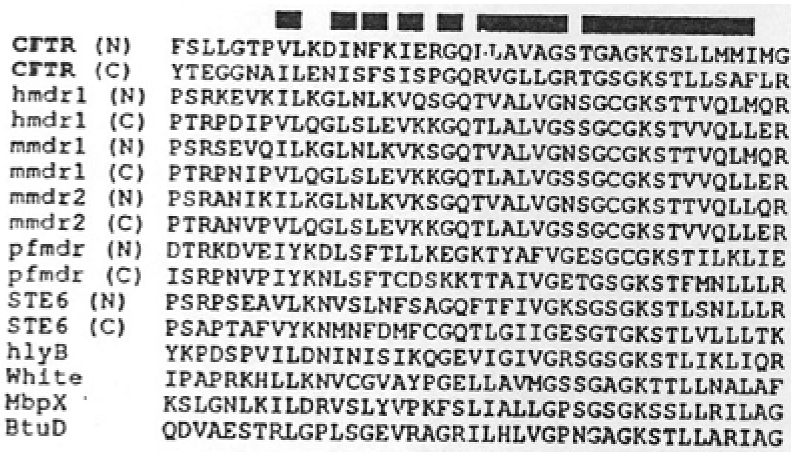
Algorithms in Bioinformatics: A Practical Introduction

Sequence Similarity

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)  YTEGGNAILENISFSISPGQRVGLLGRTGSGKSTLLSAFLR
hmdr1  (N)  PSRKEVKILKGLNLKVQSGQTVALVGNSGCGKSTTVQLMQR
hmdr1  (C)  PTRPDIPVLQGLSLEVKKGQTLALVGSSGCGKSTVVQLLER
mmdr1  (N)  PSRSEVQILKGLNLKVKSGQTVALVGNSGCGKSTTVQLMQR
mmdr1  (C)  PTRPNIPVLQGLSLEVKKGQTLALVGSSGCGKSTVVQLLER
mmdr2  (N)  PSRANIKILKGLNLKVKSGQTVALVGNSGCGKSTTVQLLQR
mmdr2  (C)  PTRANVPVLQGLSLEVKKGQTLALVGSSGCGKSTVVQLLER
pfmdr  (N)  DTRKDVEIYKDLSFTLLKEGKTYAFVGESGCGKSTILKLIE
pfmdr  (C)  ISRPNVPIYKNLSFTCDSKKTTAIVGETGSGKSTFMNLLLR
STE6   (N)  PSRPSEAVLKNVSLNFSAGQFTFIVGKSGSGKSTLSNLLLR
STE6   (C)  PSAPTAFVYKNMNFDMFCGQTLGIIGESGTGKSTLVLLLTK
hlyB        YKPDSPVILDNINISIKQGEVIGIVGRSGSGKSTLIKLIQR
White       IPAPRKHLLKNVCGVAYPGELLAVMGSSGAGKTTLLNALAF
MbpX        KSLGNLKILDRVSLYVPKFSLIALLGPSGSGKSSLLRILAG
BtuD        QDVAESTRLGPLSGEVRAGRILHLVGPNGAGKSTLLARIAG
```

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.

Applications of sequence comparison

- Inferring the biological function of gene (or RNA or protein)
  - When two genes look similar, we conjecture that both genes have similar function
- Finding the evolution distance between two species
  - Evolution modifies the DNA of species. By measuring the similarity of their genome, we know their evolution distance
- Helping genome assembly
  - Based on the overlapping information of a huge amount of short DNA pieces, Human genome project reconstructs the whole genome. The overlapping information is done by sequence comparison.
- Finding common subsequences in two genomes
- Finding repeats within a genome
- … many many other applications

Outline

- String alignment problem (Global alignment)
  - Needleman-Wunsch algorithm
  - Reduce time
  - Reduce space
- Local alignment
  - Smith-Waterman algorithm
- Semi-global alignment
- Gap penalty
  - General gap function
  - Affline gap function
  - Convex gap function
- Scoring function

String Edit

- Given two strings A and B, edit A to B with the minimum number of edit operations:
  - Replace a letter with another letter
  - Insert a letter
  - Delete a letter
- E.g.
  -
```
A = interestingly      _i__nterestingly
B = bioinformatics     bioinformatics__
                       1011011011001111
```
  - Edit distance = 11

String edit problem

- Instead of minimizing the number of edge operations, we can associate a cost function to the operations and minimize the total cost. Such cost is called edit distance.
- For the previous example, the cost function is as follows:

  -
    ```
    A= _i__nterestingly
    B= bioinformatics__
       1011011011001111
    ```
  - Edit distance = 11

	_	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

String alignment problem

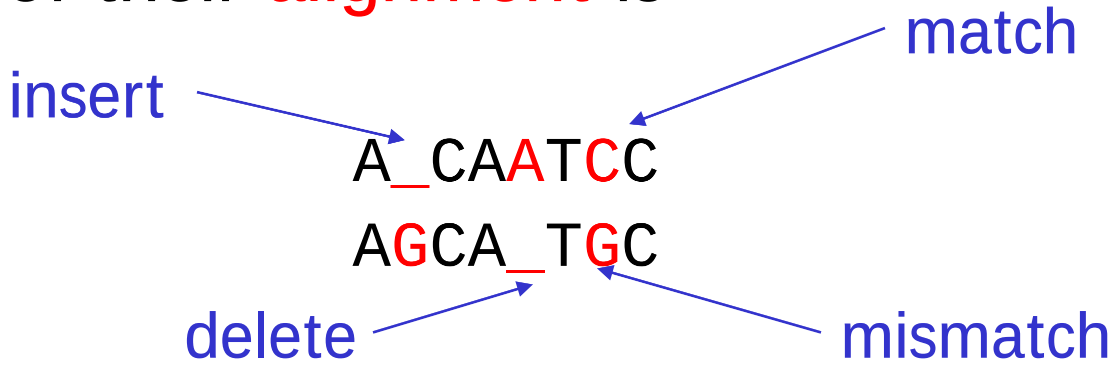
- Instead of using string edit, in computational biology, people like to use string alignment.
- We use similarity function, instead of cost function, to evaluate the goodness of the alignment.
- E.g. of similarity function: match – 2, mismatch, insert, delete – -1.

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

	_	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

String alignment

- Consider two strings ACAATCC and AGCATGC.
- One of their alignment is



```
A_CAATCC
AGCA_TGC
```

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

String alignment problem

- The alignment has similarity score 7

```
A_CAATCC
AGCA_TGC
```

- Note that the above alignment has the maximum score.
- Such alignment is called optimal alignment.

- String alignment problem tries to find the alignment with the maximum similarity score!
- String alignment problem is also called global alignment problem

Similarity vs. Distance (II)

- Lemma: String alignment problem and string edit distance are dual problems
- Proof: Exercise

- Below, we only study string alignment!

Needleman-Wunsch algorithm (I)

- Consider two strings S[1..n] and T[1..m].
- Define V(i, j) be the score of the optimal alignment between S[1..i] and T[1..j]
- Basis:
  - $V(0, 0) = 0$
  - $V(0, j) = V(0, j-1) + \delta(\_, T[j])$
    - Insert j times
  - $V(i, 0) = V(i-1, 0) + \delta(S[i], \_)$
    - Delete i times

Needleman-Wunsch algorithm (II)

- Recurrence: For i>0, j>0
  - $$V(i,j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{Match/mismatch} \\ V(i-1, j) + \delta(S[i], \_) & \text{Delete} \\ V(i, j-1) + \delta(\_, T[j]) & \text{Insert} \end{cases}$$
- In the alignment, the last pair must be either match/mismatch, delete, or insert.

```
xxx…xx          xxx…xx          xxx…x_
     |               |               |
yyy…yy          yyy…y_          yyy…yy
match/mismatch  delete          insert
```

Example (I)

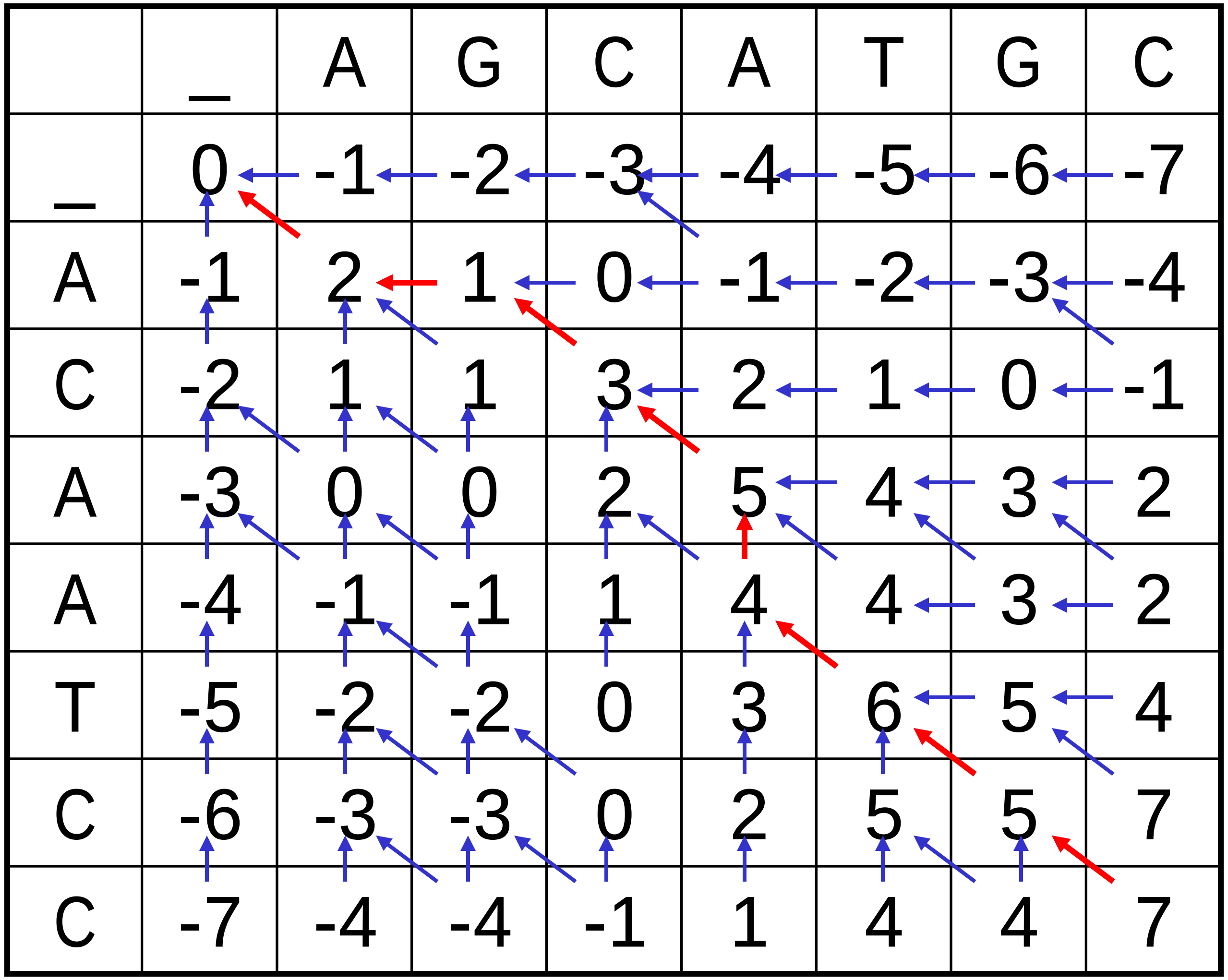
	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1							
C	-2							
A	-3							
A	-4							
T	-5							
C	-6							
C	-7							

Example (II)

	_	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)

A_CAATCC
AGCA_TGC



	_	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)

Problem on Speed (I)

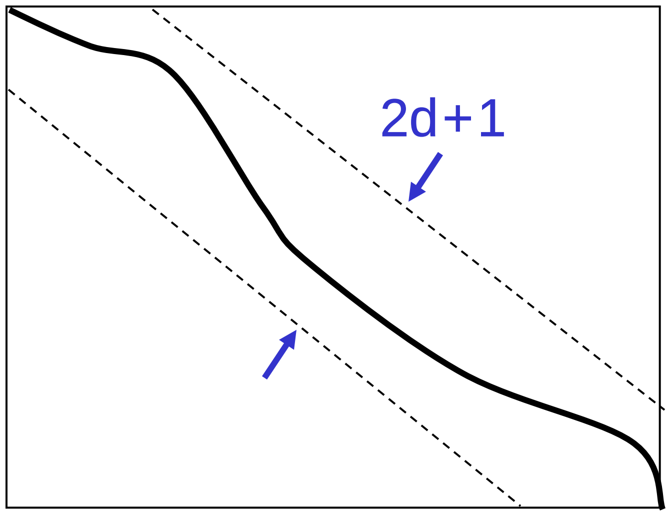
- Aho, Hirschberg, Ullman 1976
  - If we can only compare whether two symbols are equal or not, the string alignment problem can be solved in $\Omega(nm)$ time.
- Hirschberg 1978
  - If symbols are ordered and can be compared, the string alignment problem can be solved in $\Omega(n \log n)$ time.
- Masek 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.
  - $0 \le d \le n+m$
- If d is smaller than n+m, can we get a better algorithm? Yes!

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).

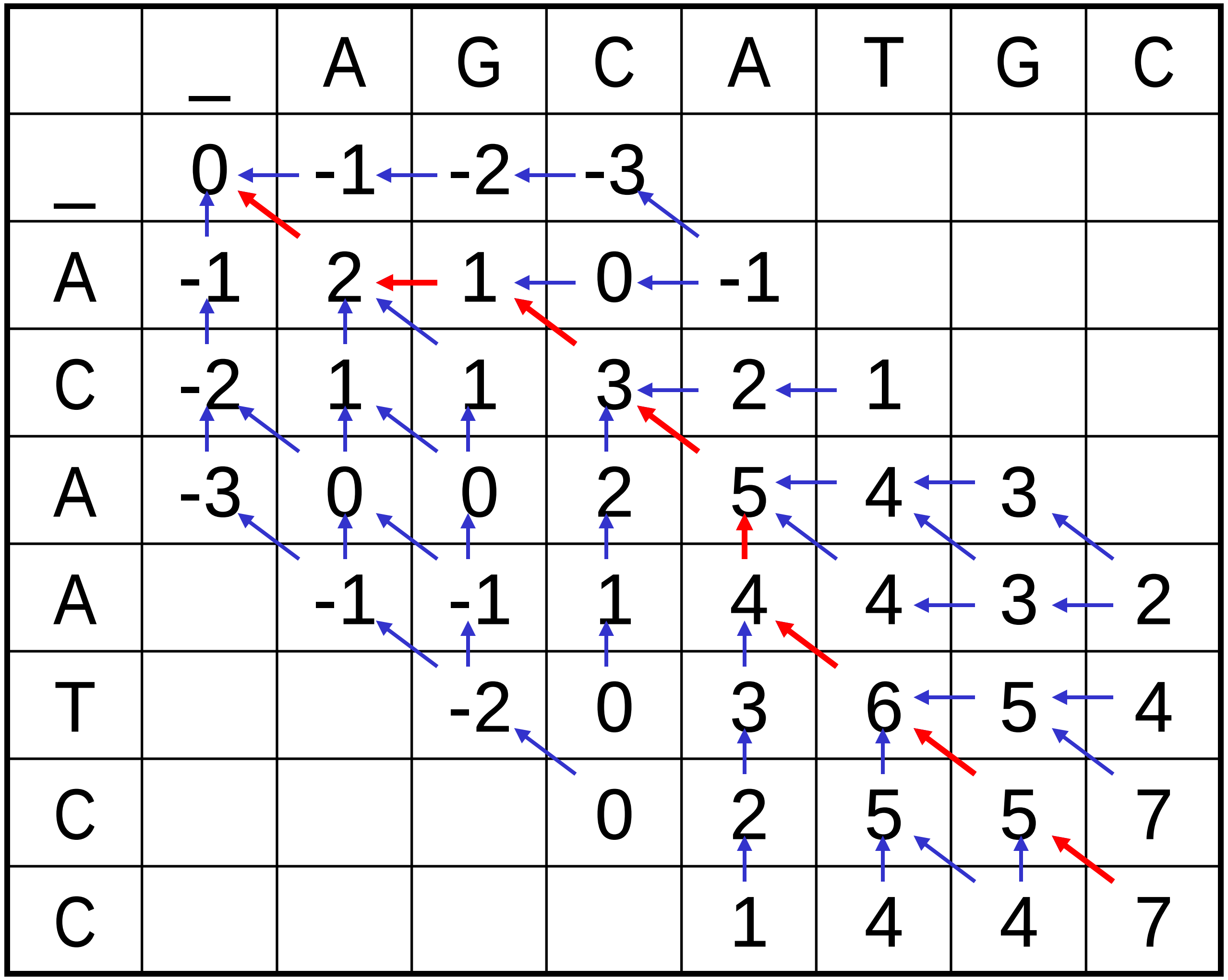


Example

- d=3

  A_CAATCC

  AGCA_TGC



	_	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

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

- In the pervious example, observe that the table can be filled in row by row.
- Thus, if we did not need to backtrack, space complexity = O(min(n, m))

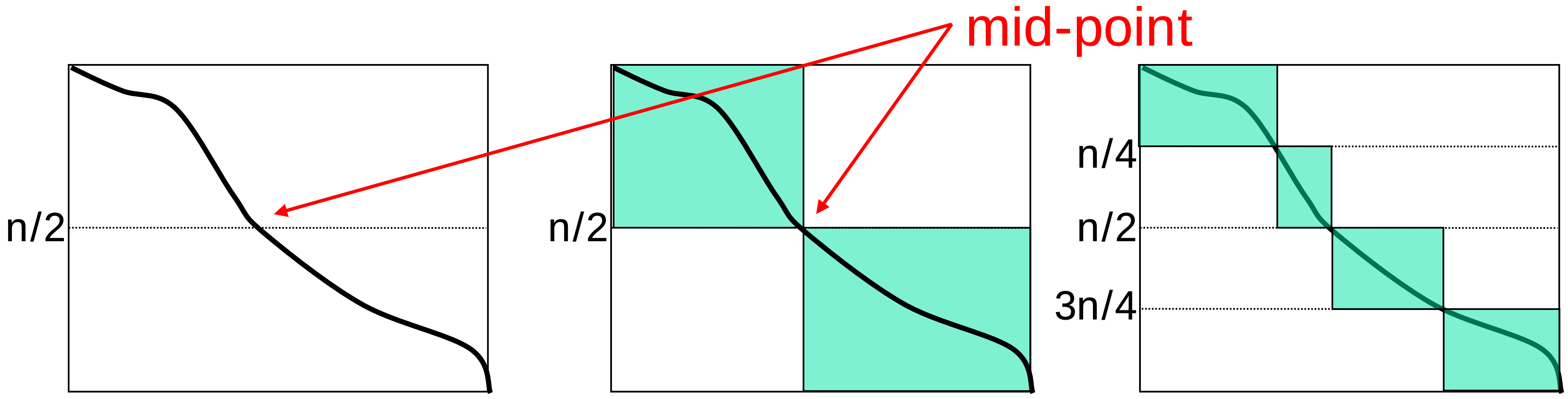
Example

	_	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!
- In general, we only need to keep two rows.

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

- Yes. Idea: By recursion!
  1. Based on the cost-only algorithm, find the mid-point of the alignment!
  2. Divide the problem into two halves.
  3. Recursively deduce the alignments for the two halves.



How to find the mid-point

Note:

$$V(S[1..n],T[1..m]) =$$
$$\max_{0\le j\le m}\left\{V(S[1..\tfrac{n}{2}],T[1..j])+V(S[\tfrac{n}{2}+1..n],T[j+1..m])\right\}$$

1. Do cost-only dynamic programming for the first half.
   - Then, we find V(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 V(S[n/2+1..n], T[j+1..m]) for all j
3. Determine j which maximizes the above sum!

Example (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 (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 (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						🟩	🟩	🟩	🟩
_						🟩	🟩	🟩	🟩

Time Analysis

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

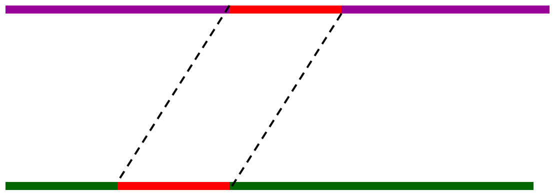
Space analysis

- 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



- Given two long DNAs, both of them contain the same gene or closely related gene.
  - Can we identify the gene?
- Local alignment problem:

  Given two strings S[1..n] and T[1..m],

  among all substrings of S and T,

  find substrings A of S and B of T whose global alignment has the highest score

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.
  - 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≥1
- X is a prefix of S[1..n] if X=S[1..k] for some k≤n
- E.g.
  - Consider S[1..7] = ACCGATT
  - ACC is a prefix of S, GATT is a suffix of S
  - Empty string is both prefix and suffix of S

Dynamic programming for local alignment problem

- Define V(i, j) be the maximum score of the global alignment of A and B over
  - all suffixes A of S[1..i] and
  - all suffixes B of T[1..j]
- Note:
  - all suffixes of S[1..i] = all substrings in S end at i
  - {all suffixes of S[1..i]|i=1,2,…,n} = all substrings of S
- Then, score of local alignment is
  - $\max_{i,j} V(i,j)$

Smith-Waterman algorithm

- Basis:
  - V(i, 0) = V(0, j) = 0
- Recursion for i>0 and j>0:
  - $$V(i,j) = \max\begin{cases} 0 & \text{Align empty strings} \\ V(i-1, j-1) + \delta(S[i], T[j]) & \text{Match/mismatch} \\ V(i-1, j) + \delta(S[i], \_) & \text{Delete} \\ V(i, j-1) + \delta(\_, T[j]) & \text{Insert} \end{cases}$$

Example (I)

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

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

Example (II)

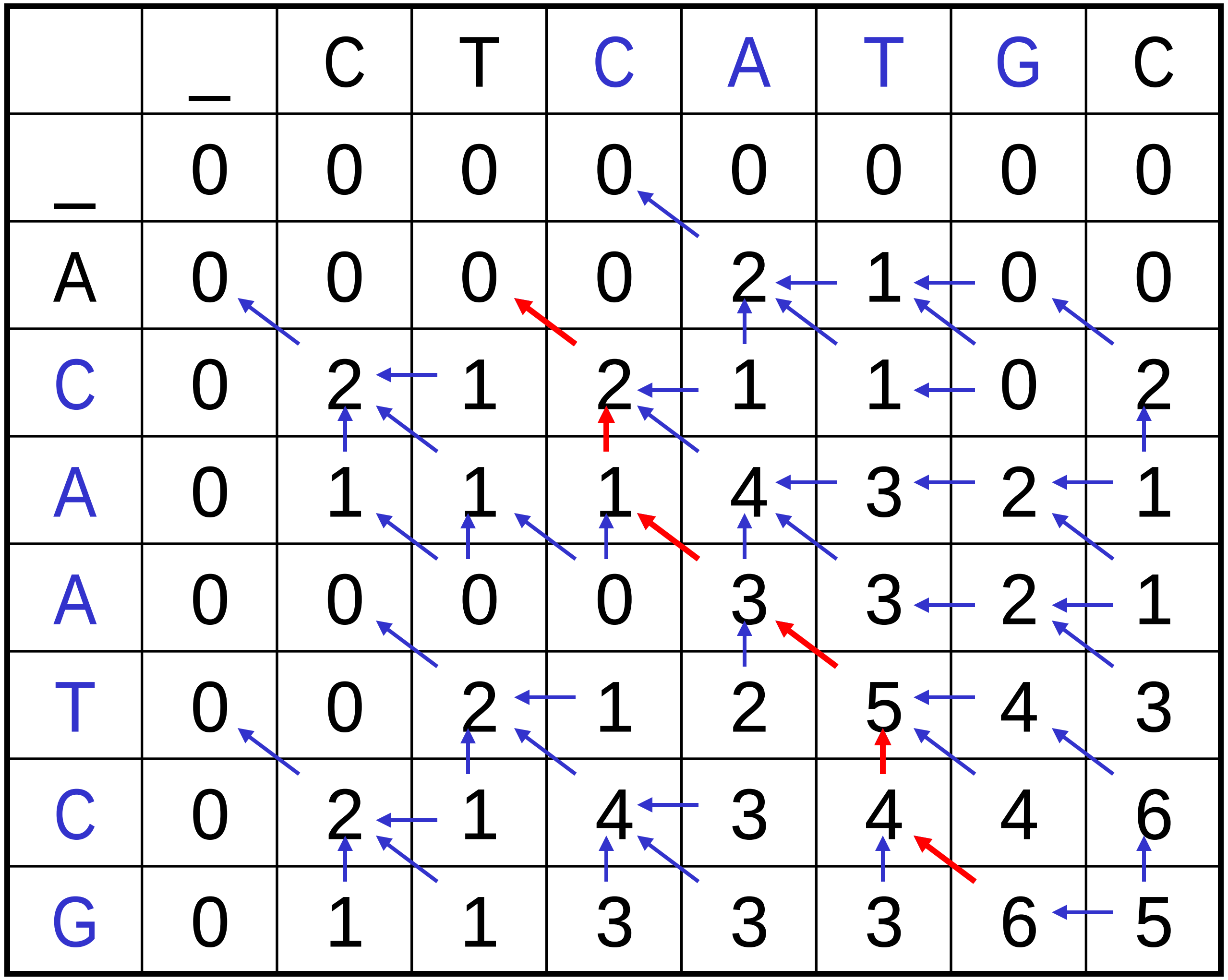
- Score for match = 2
- Score for insert, delete, mismatch = -1

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

Example (III)

CAATCG

C_AT_G



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

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)

More on local alignment

- Similar to global alignment,
  - we can reduce the space requirement
- Exercise!

Semi-global alignment

- Semi-global alignment ignores some end spaces
- Example 1: ignoring beginning and ending spaces of the second sequence.

  -
    ```
    ATCCGAA_CATCCAATCGAAGC
    ______AGCATGCAAT______
    ```
  - The score of below alignment is 14
    - 8 matches (score=16), 1 delete (score=-1), 1 mismatch (score=-1)
  - This alignment can be used to locate gene in a prokaryotic genome

Semi-global alignment

- Example 2: ignoring beginning spaces of the 1st sequence and ending spaces of the 2nd sequence

  -
    ```
    _________ACCTCACGATCCGA
    TCAACGATCACCGCA________
    ```
  - The score of above alignment is 9
    - 5 matches (score=10), 1 mismatch (score=-1)
  - This alignment can be used to find the common region of two overlapping sequences

How to compute semi-global alignment?

- In general, we can forgive spaces
  - in the beginning or ending of S[1..n]
  - in the beginning or ending of T[1..m]
- Semi-global alignment can be computed using the dynamic programming for global alignment with some small changes.
- Below table summaries the changes

Spaces that are not charged	**Action**
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_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 large substring. Such kind of mutation may be as likely as insertion/deletion of a single base.
  - Recall that mRNA misses the introns. When aligning mRNA with its gene, the penalty should not be proportional to the length of the gaps.

General gap penalty (I)

- Definition: g(q) is denoted as the penalty of a gap of length q
- Global alignment of S[1..n] and T[1..m]:
  - Denote V(i, j) be the score for global alignment between S[1..i] and T[1..j].
  - Base cases:
    - V(0, 0) = 0
    - V(0, j) = -g(j)
    - V(i, 0) = -g(i)

General gap penalty (II)

- Recurrence: for i>0 and j>0,
  - $$V(i,j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{Match/mismatch} \\ \max\limits_{0 \le k \le j-1} \{V(i,k) - g(j-k)\} & \text{Insert T[k+1..j]} \\ \max\limits_{0 \le k \le i-1} \{V(k,j) - g(i-k)\} & \text{Delete S[k+1..i]} \end{cases}$$

Analysis

- We need to fill in all entries in the $n \times m$ table.
- Each entry can be computed in $O(n+m)$ time.
- Time complexity = $O(n^2m + nm^2)$
- Space complexity = $O(nm)$

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 $g(q) = h+qs$

How to compute alignment using affline gap model?

- By the previous dynamic programming, the problem can be solved in $O(n^2m+nm^2)$ time.
- Can we do faster?

- Yes!
- Idea: Have a refined dynamic programming!

Dynamic programming solution (I)

- Recall V(i, j) is the score of a global optimal alignment between S[1..i] and T[1..j]
- Decompose V(i,j) into three cases:
  - G(i, j) is the score of a global optimal alignment between S[1..i] and T[1..j] with S[i] aligns 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] aligns with a space
  - E(i, j) is the score of a global optimal alignment between S[1..i] and T[1..j] with a space aligns with T[j]

```
xxx…xx        xxx…xx        xxx…x_
     |             |             |
yyy…yy        yyy…y_        yyy…yy
 G(i,j)        F(i,j)        E(i,j)
```

Dynamic programming solution (II)

- Basis:
  - V(0, 0) = 0
  - V(i, 0) = -h-is; V(0, j) = -h-js
  - E(i, 0) = -∞
  - F(0, j) = -∞

Dynamic programming solution (III)

- Recurrence:
  - V(i, j) = max { G(i, j), F(i, j), E(i, j) }

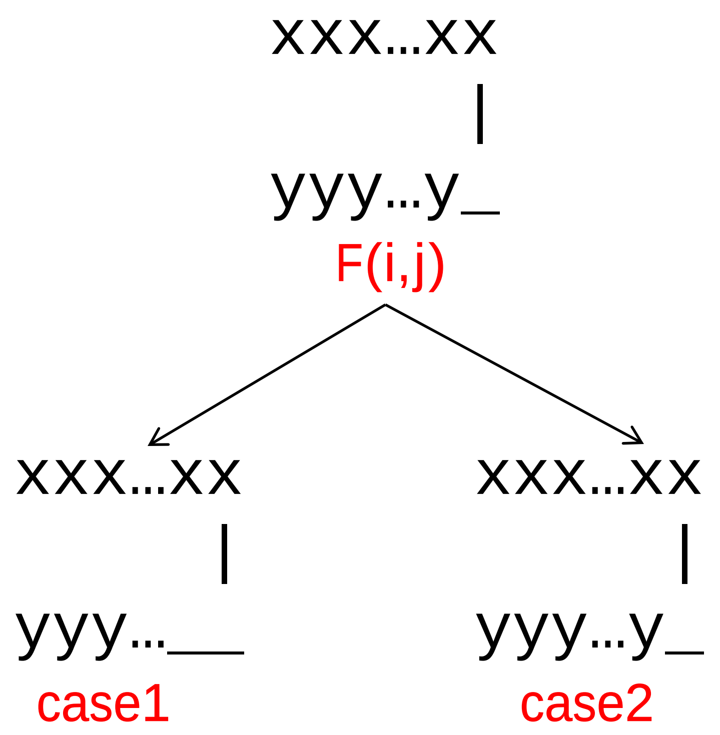
```
xxx…xx          xxx…xx        xxx…x_
     |               |             |
yyy…yy          yyy…y_        yyy…yy
 G(i,j)          F(i,j)        E(i,j)
```

  - G(i, j) = V(i-1, j-1) + δ(S[i], T[j])

```
xxx…xx
     |
yyy…yy
 G(i,j)
```

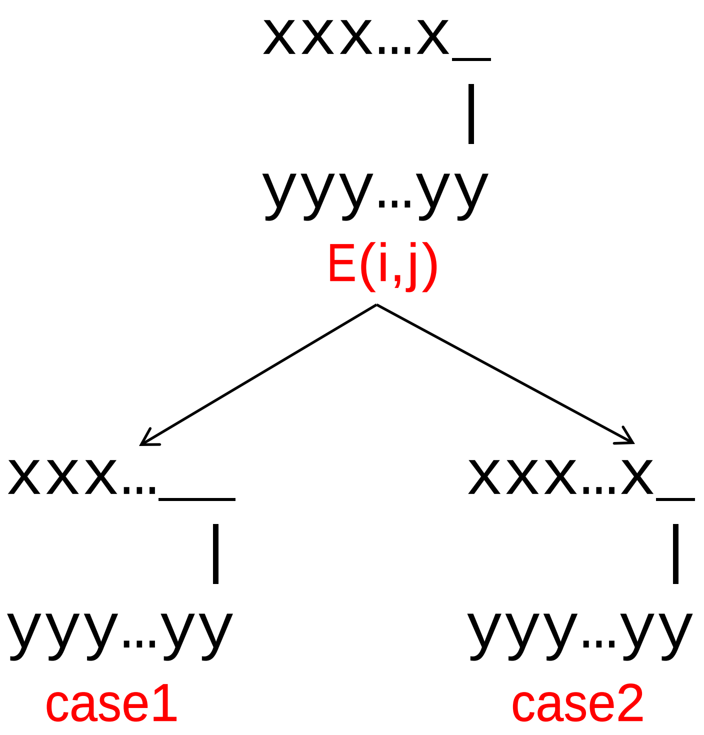
Dynamic programming solution (IV)

- Recurrence:
  - F(i, j) = max { F(i-1, j)–s, V(i-1, j)–h–s }



Dynamic programming solution (V)

- Recurrence:
  - E(i, j) = max { E(i, j-1)–s, V(i, j-1)–h–s }



Summary of the recurrences

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

Analysis

- We need to fill in 4 tables, each is of size $n \times m$.
- Each entry can be computed in O(1) time.
- Time complexity = O(nm)
- Space complexity = O(nm)

Is affine gap penalty good?

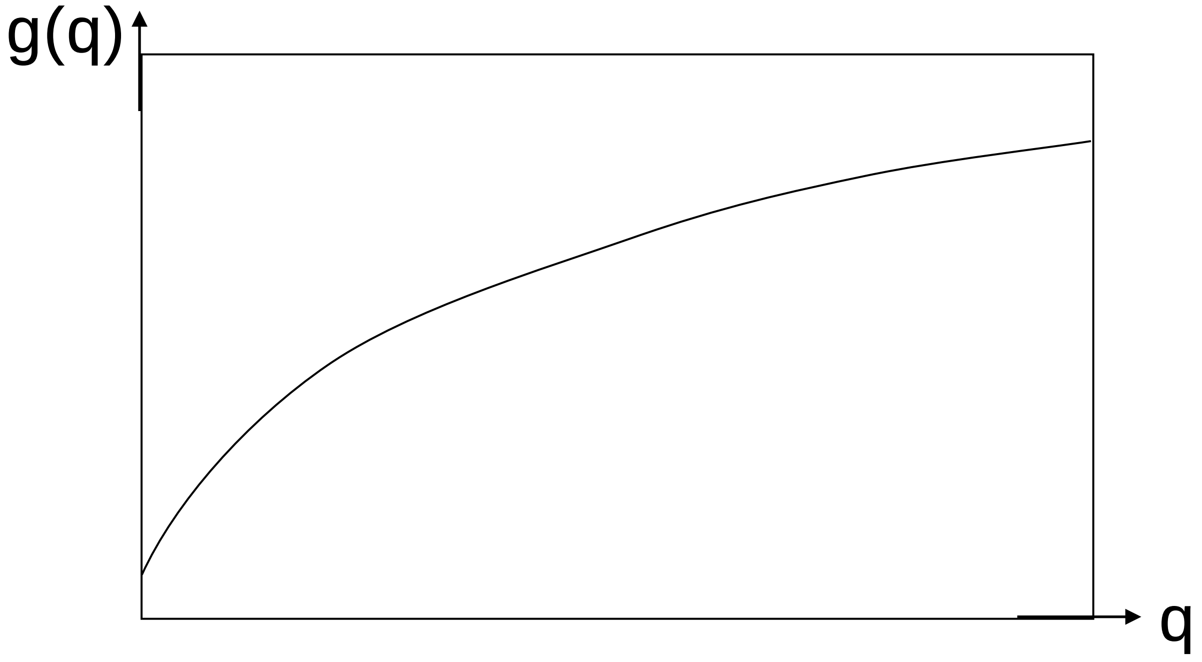
- Affine gap penalty fails to approximate some real biological mechanisms.
  - For example, affine gap penalty is not in favor of long gaps.

- People suggested other non-affine gap penalty functions. All those functions try to ensure:
  - The penalty incurred by additional space in a gap decrease as the gap gets longer.
  - Example: the logarithmic gap penalty $g(q) = a \log q + b$

Convex gap penalty function

- A convex gap penalty function g(q) is a non-negative increasing function such that

  $g(q+1) - g(q) \leq g(q) - g(q-1)$ for all $q \geq 1$



Alignment with convex gap penalty

- By dynamic programming, the alignment can be found in O(nm^2+n^{2}m) time.

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

$$A(i,j)=\max_{0\le k\le j-1}\{V(i,k)-g(j-k)\}$$

$$B(i,j)=\max_{0\le k\le i-1}\{V(k,j)-g(i-k)\}$$

- If the gap penalty function g() is convex, can we improve the running time?

Alignment with Convex gap penalty

- Given A() and B(), V(i,j) can be computed in O(nm) time.
- Below, for convex gap penalty, we show that
  - A(i, 1), …, A(i, m) can be computed in O(m log m) time.
  - Similarly, B(1, j), …, B(n, j) can be computed in O(n log n) time.
- In total, all entries V(i, j) can be filled in O(nm log(nm)) time.

Subproblem

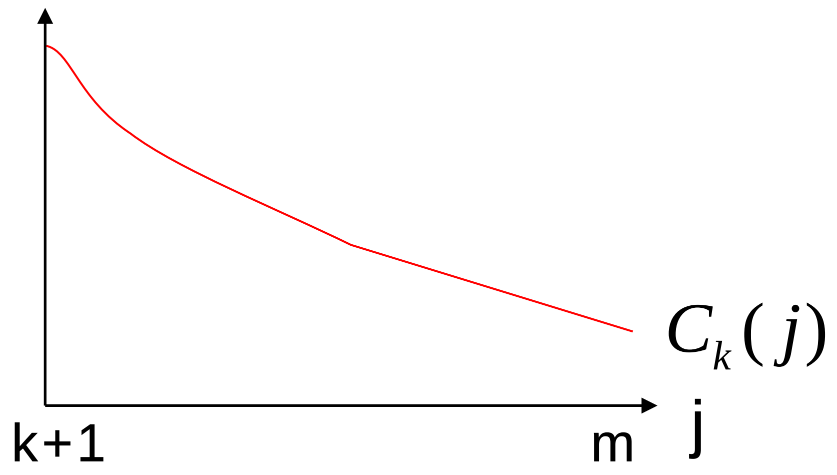
- For a fixed i, let
  - E(j) = A(i, j) and $C_k(j)$ = V(i,k) – g(j-k).
- Recurrence of A(i, j) can be rewritten as

$$E(j) = \max_{0 \le k < j} \{C_k(j)\}$$

- By dynamic programming, E(1), …, E(m) can be computed in $O(m^2)$ time.
- We show that E(1), …, E(m) can be computed in O(m log m) time.

Properties of $C_k(j)$

- $C_k(j)$ is a decreasing function.
- As j increases, the decreasing rate of $C_k(j)$ is getting slower and slower.

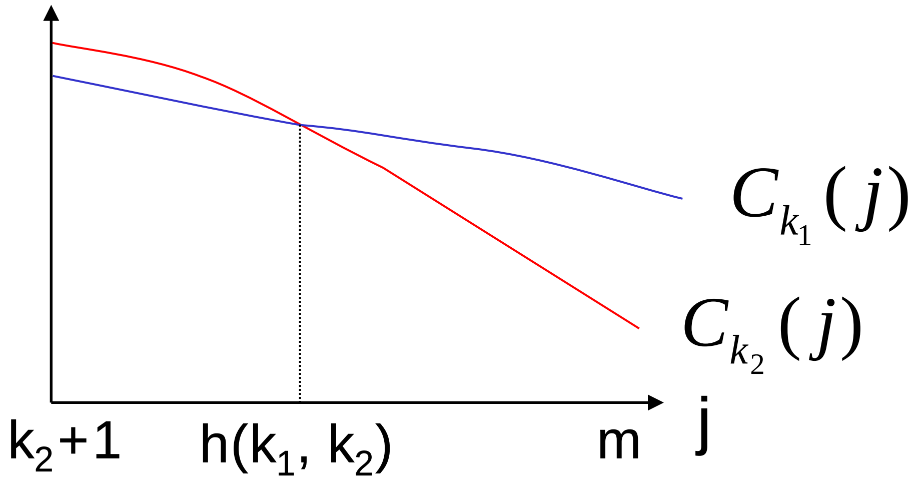


Lemma

Note: for a fixed k, $C_k(j')$ is a decreasing function

- For any $k_1 < k_2$, let $h(k_1, k_2) = \arg\min_{k_2 < j \le m} \{C_{k_1}(j) \ge C_{k_2}(j)\}$
- We have
  - $j < h(k_1, k_2)$ if and only if $C_{k_1}(j) < C_{k_2}(j)$.

The two curves intersect at most one!



- $h(k_1, k_2)$ can be found in O(log m) time by binary search.

Proof of the lemma

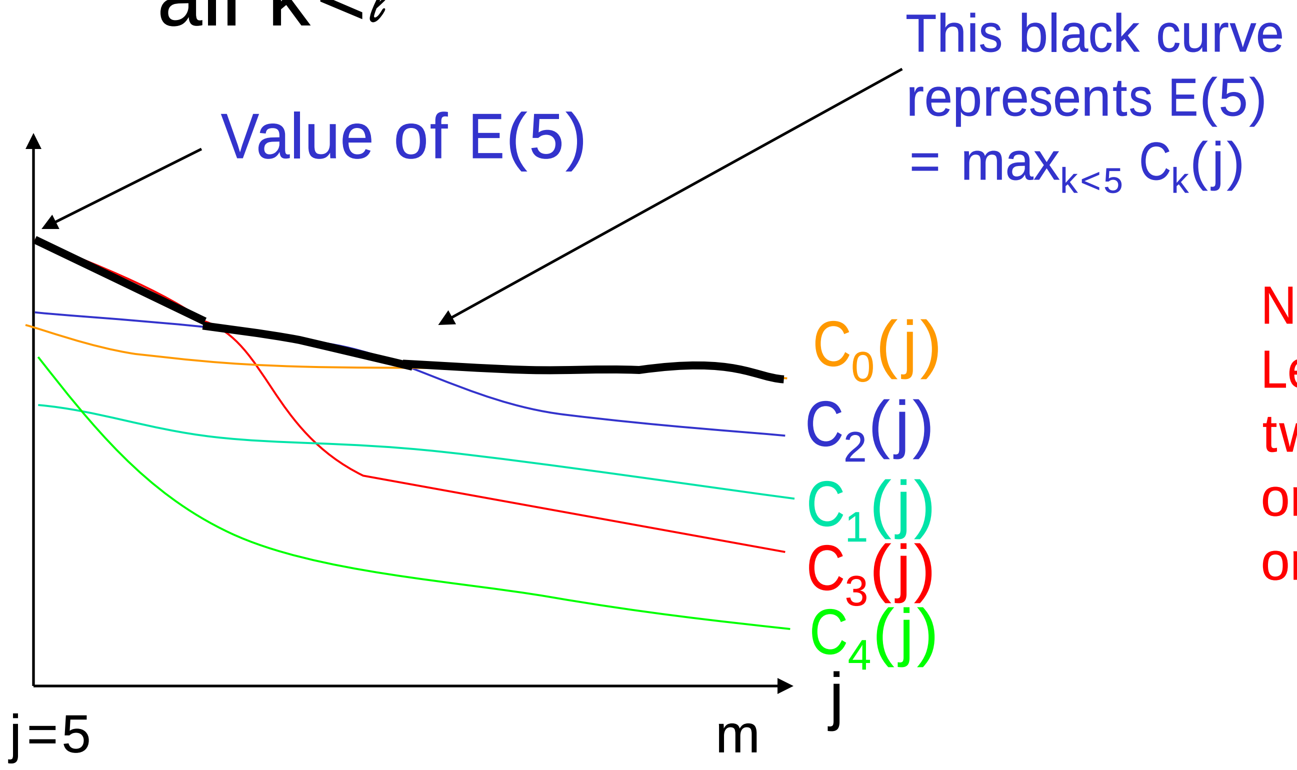
1. If $k_2 < j < h(k_1, k_2)$, by definition, $C_{k1}(j) < C_{k2}(j)$.
2. Otherwise, we show that $C_{k1}(j) \geq C_{k2}(j)$ for $h(k_1, k_2) \leq j \leq m$ by induction.

- When $j=h(k_1,k_2)$, by definition, $C_{k1}(j) \geq C_{k2}(j)$.
- Suppose $C_{k1}(j) \geq C_{k2}(j)$ for some j. Then,

$$\begin{aligned}
C_{k_1}(j+1) &= C_{k_1}(j) - g(j+1-k_1) + g(j-k_1) \\
&\geq C_{k_2}(j) - g(j+1-k_1) + g(j-k_1) && \text{since } C_{k_1}(j) \geq C_{k_2}(j) \\
&\geq C_{k_2}(j) - g(j+1-k_2) + g(j-k_2) && \text{since } g(q) \text{ is convex} \\
&= C_{k_2}(j+1)
\end{aligned}$$

Frontier of all curves

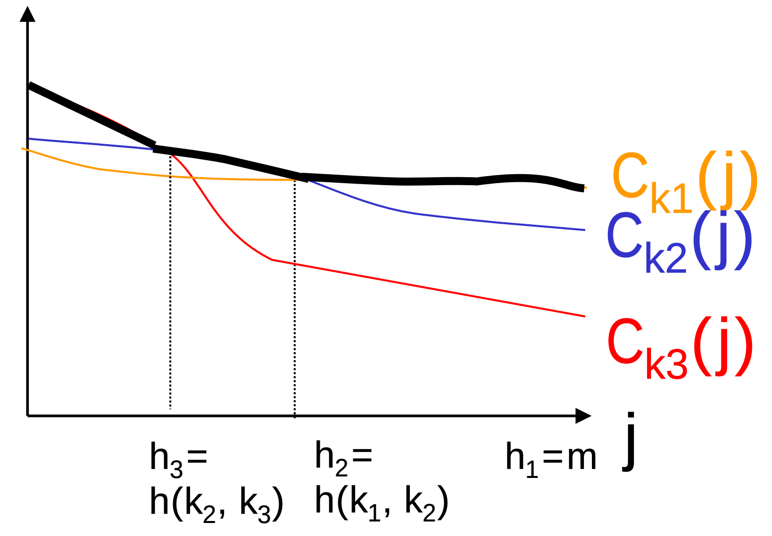
- For a fixed j, consider curves $C_k(j)$ for all $k<\ell$



Note: By Lemma, any two curves can only intersect at one point

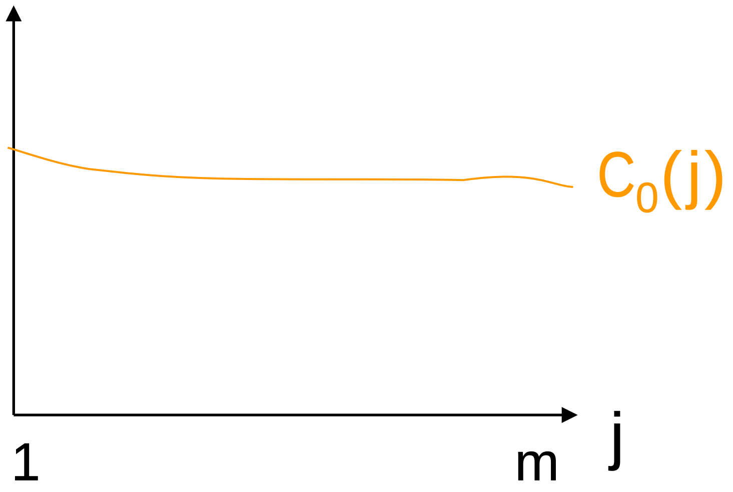
Frontier of all curves

- Thus, for a fixed j, the black curve can be represented by $(k_{top}, h_{top}), (k_{top-1}, h_{top-1}), \ldots, (k_1, h_1)$
- Note that
  $k_1 < \ldots < k_{top} < j < h_{top} < \ldots < h_1$ (by default, $h_1 = m$)
- In this algorithm, (k_x, h_x) are stored in a stack with (k_{top}, h_{top}) at the top of the stack!



$\max_{k<1} C_k(j)$

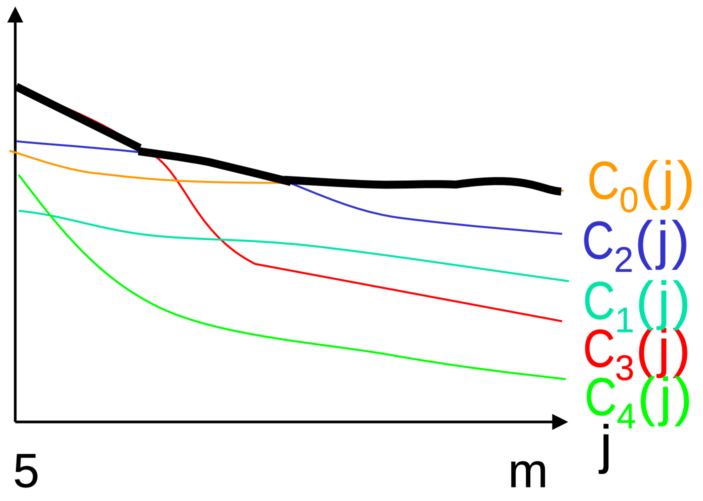
- For $\ell = 1$,
  - The set of curves $\{C_k(j) \mid k<\ell\}$ contains only curve $C_0(j)$. Thus,
    - $\max_{k<\ell} C_k(j) = C_0(j)$.
  - Thus, $\max_{k<\ell} C_k(j)$ can be represented by $(k_0=0, h_0=m)$

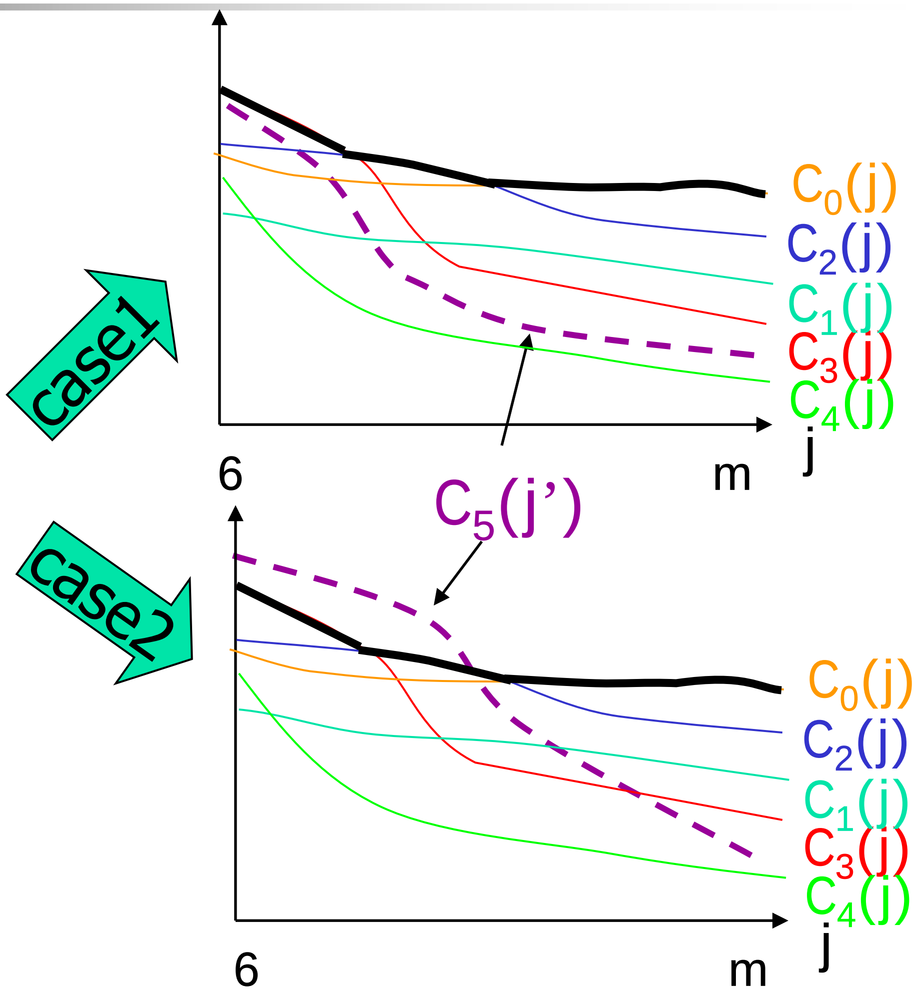


$\max_{k<\ell} C_k(j)$ for $\ell>1$

- For a particular j, suppose the curve $\max_{k<\ell} C_k(j)$ is represented by $(k_{top}, h_{top}), \ldots, (k_0, h_0)$.
- How can we get the curve $\max_{k<\ell+1} C_k(j)$?

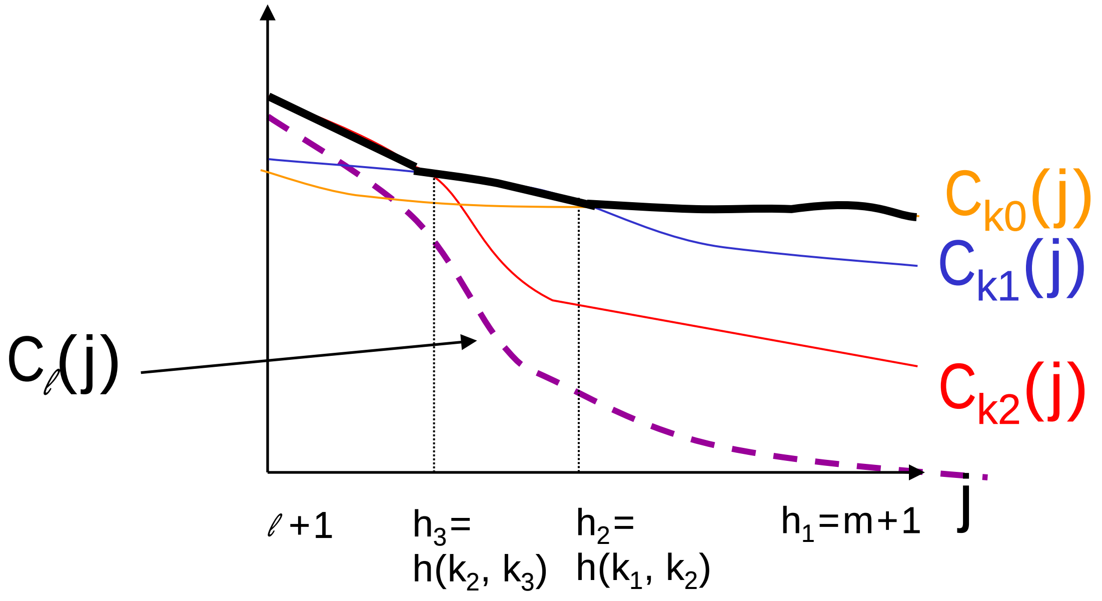
Frontier for $\max_{k<\ell+1} C_k(j)$





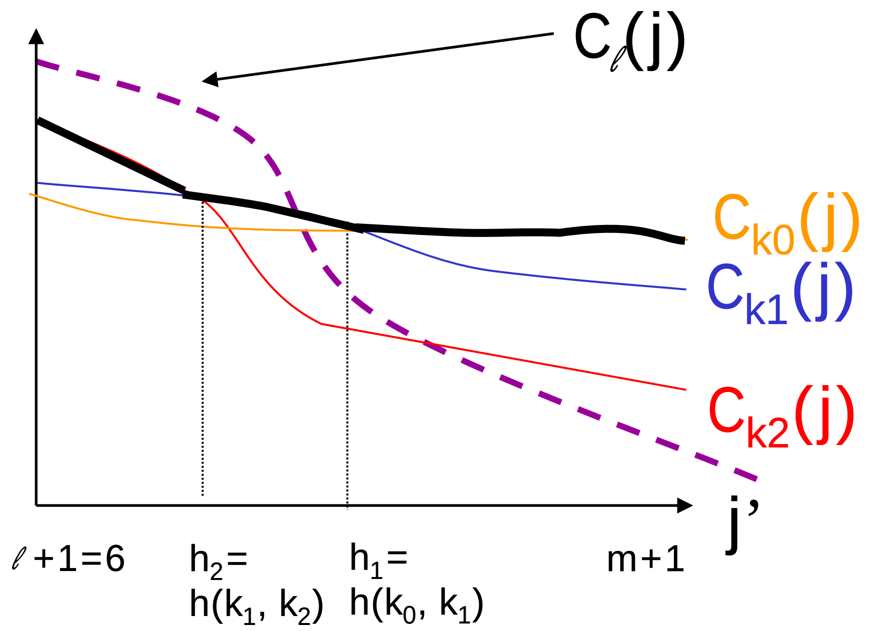
Frontier (case 1)

- If $C_\ell(\ell+1) \leq C_{ktop}(\ell+1)$,
  - the curve $C_\ell(j)$ cannot cross $C_{ktop}(j)$ and it must be below $C_{ktop}(j)$.
- Thus, the black curve for $\max_{k<\ell+1} C_k(j)$ is the same as that for $\max_{k<\ell} C_k(j)$!



Frontier (case 2)

- If C(j, j+1)>C(k_{top}, j+1),
  - the curve $\max_{k<j+1}$ C(k, j') is different from the curve $\max_{k<j}$ C(k, j'). We need to update it.



Algorithm

```
Push $(0, m)$ onto stack S.
$E[1] = C_{ktop}(1)$;
For $\ell = 1$ to $m-1$ {
    if $C_\ell(\ell+1) > C_{ktop}(\ell+1)$ then {
        While $S \neq \Phi$ and $C_\ell(h_{top}-1) > C_{ktop}(h_{top}-1)$ do
            Pop S;
        if $S = \Phi$ then
            Push $(\ell, m+1)$ onto S
        else
            Push $(\ell, h(k_{top}, \ell))$;
    }
    $E[\ell] = C_{ktop}(\ell)$;
}
```

Analysis

- For every ℓ, we will push at most one pair onto the stack S.
    - Thus, we push at most m pairs onto the stack S.
    - Also, we can only pop at most m pairs out of the stack S
- The h value of each pair can be computed in O(log m) time by binary search.
- The total time is O(m log m).

Scoring function

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

Scoring function for DNA

- For DNA, since we only have 4 nucleotides, the score function is simple.
  - BLAST matrix
  - Transition Transversion matrix: give mild penalty for replacing purine by purine. Similar for replacing pyrimadine by pyrimadine!

	A	C	G	T
A	5	-4	-4	-4
C	-4	5	-4	-4
G	-4	-4	5	-4
T	-4	-4	-4	5

BLAST 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 Matrix

Scoring function for Protein

- Commonly, it is devised based on two criteria:
  - Chemical/physical similarity
  - Observed substitution frequencies

Scoring function for protein using physical/chemical properties

- Idea: an amino acid is more likely to be substituted by another if they have similar property
- See Karlin and Ghandour (1985, PNAS 82:8597)
- The score matrices can be derived based on hydrophobicity, charge, electronegativity, and size
- E.g. we give higher score for substituting nonpolar amino acid to another nonpolar amino acid

Scoring function for protein based on statistical model

- Most often used approaches

- Two popular matrices:
  - Point Accepted Mutation (PAM) matrix
  - BLOSUM
- Both methods define the score as the log-odds ratio between the observed substitution rate and the actual substitution rate

Point Accepted Mutation (PAM)

- PAM was developed by Dayhoff (1978).

- A point mutation means substituting 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.
- Two sequence S_1 and S_2 are said to be 1 PAM diverged if a series of accepted point mutation can convert S_1 to S_2 with an average of 1 accepted point mutation per 100 residues

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
    ```

PAM matrix by example (II)

- Build the phylogenetic tree for the sequences

$$\begin{tikzcd}
 & & & \text{IACGCTAFK} \arrow[dll, "A\to G"'] \arrow[drr, "I\to L"] & & & \\
 & \text{IGCGCTAFK} \arrow[dl, "A\to G"'] \arrow[dr, "A\to L"] & & & & \text{LACGCTAFK} \arrow[dl, "C\to S"'] \arrow[dr, "G\to A"] & \\
\text{IGCGCTGFK} & & \text{IGCGCTLFK} & & \text{LASGCTAFK} & & \text{LACACTAFK}
\end{tikzcd}$$

PAM-1 matrix

$$\delta(a,b) = \log_{10} \frac{O_{a,b}}{E_{a,b}}$$

where $O_{a,b}$ and $E_{a,b}$ are the observed frequency and the expected frequency.

- Since PAM-1 assume 1 mutation per 100 residues,
  - $O_{a,a}$ = 99/100.
- For $a \neq b$,
  - $O_{a,b} = F_{a,b} / (100\ \Sigma_x \Sigma_y F_{x,y})$ where $F_{a,b}$ is the frequency $F_{a,b}$ of substituting a by b or b by a.
- $E_{a,b} = f_a * f_b$ where f_a is the no. of a divided by total residues

- E.g., $F_{A,G}$ = 3, $F_{A,L}$=1. $f_A = f_G$ = 10/63.
  - $O_{A,G}$ = 3/(100*2*6) = 0.0025
  - $E_{A,G}$ = (10/63)(10/63) = 0.0252
  - δ(A,G) = log (0.0025 / 0.0252) = log (0.09925) = -1.0034

PAM-2 matrix

- Let $M_{a,b}$ be the probability that a is mutated to b, which equals $O_{a,b} / f_a$.

- PAM-2 matrix is created by extrapolate PAM-1 matrix.
- $M^2(a,b) = \sum_x M(a,x)M(x,b)$ is the probability that a is mutated to b after 2 mutations.

- Then, (a,b) entry of the PAM-2 matrix is
  $\log(f_a\, M^2(a,b)/f_a\, f_b) = \log(M^2(a,b)/f_b)$

PAM-n matrix

- Let $M_{a,b}$ be the probability that a is mutated to b, which equals $O_{a,b}$ / f_a.

- In general, PAM-n matrix is created by extrapolate PAM-1 matrix.
- $M^n(a,b)$ is the probability that a is mutated to b after n mutations.

- Then, (a,b) entry of the PAM-n matrix is

$$\log(f_a M^n(a,b)/f_a f_b) = \log(M^n(a,b)/f_b)$$

BLOSUM (BLOck SUbstition Matrix)

- PAM did not work well for aligning evolutionarily divergent sequences since the matrix is generated by extrapolation.
- 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 aligments are derived.
- Each block represents a conserved region of a protein family.

Extract frequencies from blocks

- From all blocks, we count the frequency f_a for each amino acid residue a.
- For any two amino acid residues a and b, we count the frequency p_{ab} of aligned pair of a and b.

- For example,

  -
    ```
    ACGCTAFKI
    GCGCTAFKI
    ACGCTAFKL
    GCGCTGFKI
    GCGCTLFKI
    ASGCTAFKL
    ACACTAFKL
    ```
- There are 7*9=63 residues, including 9's A and 10's G. Hence, f_A = 10/63, f_G = 10/63.

- There are $9\binom{7}{2}=189$ aligned residue pairs, including 23 (A,G) pairs. Hence, O_{AG} = 23 / 189.

The scoring function of BLOSUM

- For each pair of aligned residues a and b, the alignment score $\delta(a,b) = 1/\lambda \ln O_{ab}/(f_a f_b)$
  - where O_{ab} is the probability that a and b are observed to align together. f_a and f_b are the frequency of residues a and b respectively. λ is a normalization constant.
- Example: $f_A=10/63$, $f_G=10/63$, $O_{AG} = 23/189$. With $\lambda=0.347$, $\delta(A,L)=4.54$.

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 sequeneces and we get
  - AV[$A_{0.75}L_{0.25}$]AA
    VVAAL
- Then, $O_{AV} = 1 / 5$ and $O_{AL} = (0.25 + 1)/5$.

Relationship between BLOSUM and PAM

- Relationship between BLOSUM and PAM
  - BLOSUM 80 ≈ PAM 1
  - BLOSUM 62 ≈ PAM 120
  - BLOSUM 45 ≈ PAM 250
- BLOSUM 62 is the default matrix for BLAST 2.0

BLOSUM 62

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