Sequence Alignment

Young-Rae Cho, Ph.D.
Professor
Division of Software / Division of Digital Healthcare
Yonsei University – Mirae Campus

Overview

1. Backgrounds
2. Manhattan Tourist Problem
3. Longest Common Subsequence Problem
4. Edit Distance
5. Global Sequence Alignment
6. Local Sequence Alignment
7. Alignment with Gap Penalty
8. Sequence Homolog Search
9. Multiple Sequence Alignment

Homologs, Orthologs, Paralogs, and Analogs

- **Homologs**
  - similar sequence + common ancestor (divergent evolution)
  - Orthlogs: homologs in different species by species divergence
  - Paralogs: homologs in the same species by gene duplication

- **Analogs**
  - similar sequence + no common ancestor (convergent evolution)

- **How to measure sequence similarity?**

1) Counting identical letters on each position

```
A C G T T A T
  | | |     |
T C G T A C T
```

2) Inserting gaps to maximize the number of identical letters

```
A C G T T A - T
  | | |   |   |
T C G T - A C T
```

Sequence Similarity Measures

- **Measure (1)**
  - Compares the letters on the same position between two sequences
  - Not applicable to measurement of evolutionary distance

- **Measure (2)**
  - Compares the letters in the same order (even on different positions) between two sequences
  - More applicable to measurement of evolutionary distance
  - Why?
  - Example?

Sequence Alignment

- **Definition**
  - Arranging two or more DNA or protein sequences by inserting gaps to maximize their sequence similarity score
  - What is sequence similarity score?
    - the number of identical positions?
    - the sum of scores by any scoring scheme?

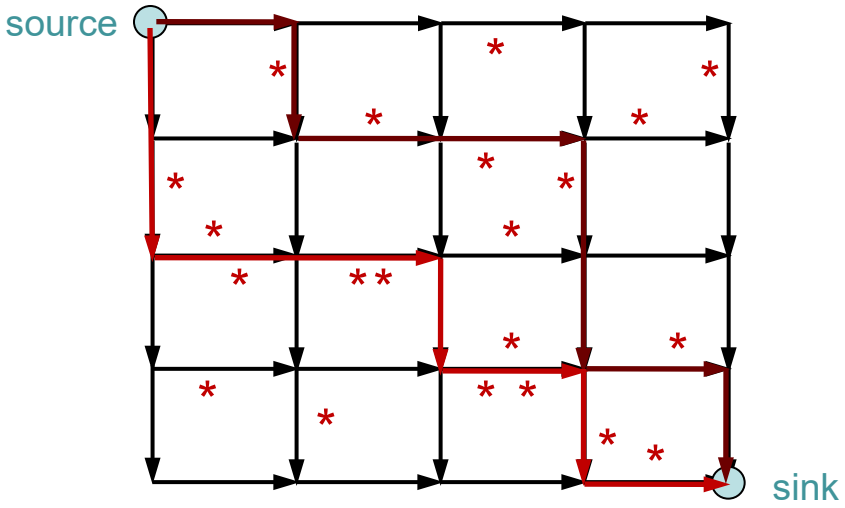
- **Applications**
  - Given gene sequences, infer their evolutionary history (phylogenetics)
  - Given gene sequences of known functions, infer the functions of newly sequenced genes
  - Given genes of known functions in one organism, infer the functions of genes in another organism

Overview

1. Backgrounds
2. Manhattan Tourist Problem
3. Longest Common Subsequence Problem
4. Edit Distance
5. Global Sequence Alignment
6. Local Sequence Alignment
7. Alignment with Gap Penalty
8. Sequence Homolog Search
9. Multiple Sequence Alignment

Manhattan Tourist Problem (MTP)

- **Problem Definition**
  - A tourist seeks a path to travel with the most attractions in Manhattan road map (grid structure)
  - Restrictions
    - A path from a source to a sink
    - A path only eastward and southward

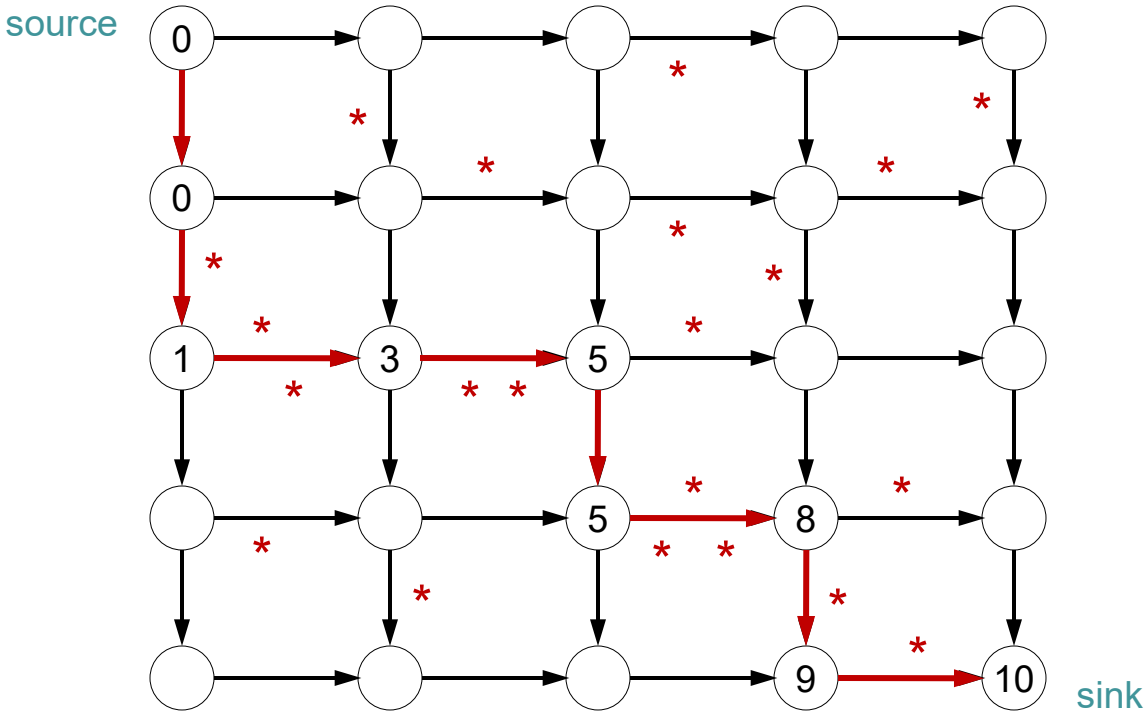


Formulation of MTP

- **Goal**
  - Finding the strongest path from a source to a sink in a weighted grid
    - The weight of an edge is defined as the number of attractions
    - The path strength is measured by summing the weights on the path
- **Input**
  - A weighted grid G with two distinct vertices, source and sink
- **Output**
  - A strongest path in G from the source to the sink

Example of MTP

- **Example**



Solving by Exhaustive Search

- **Algorithm**
  1) Enumerate all possible paths from the source to the sink
  2) Compute the path strength for every path
  3) Find the strongest path

- **Problems ?**

- **Runtime ?**

Solving by Greedy Algorithm

- **Algorithm**
  1) Start from the source
  2) Select the edge having the higher weight
  3) Repeat (2) until it reaches the sink

- **Problems ?**

- **Runtime ?**

Solving by Recursive Algorithm

- **Algorithm**

$\mathrm{MTP}(m, n)$

if $m = 0$ *and* $n = 0$

  return 0

else if $m = 0$ *and* $n \neq 0$

  return $\mathrm{MTP}(m, n-1) + w((m, n-1), (m, n))$

else if $m \neq 0$ *and* $n = 0$

  return $\mathrm{MTP}(m-1, n) + w((m-1, n), (m, n))$

else

  $x \leftarrow \mathrm{MTP}(m-1, n) + w((m-1, n), (m, n))$

  $y \leftarrow \mathrm{MTP}(m, n-1) + w((m, n-1), (m, n))$

  return $max(x, y)$

- **Problems ?**

- **Runtime ?**

Solving by Dynamic Programming

Algorithm

$\mathrm{MTP}(m, n)$

$S_{0,0} \leftarrow 0$

$for \;\; i \leftarrow 1 \;\; to \;\; m$

$\quad S_{i,0} \leftarrow S_{i-1,0} + w((i-1,0),(i,0))$

$for \;\; j \leftarrow 1 \;\; to \;\; n$

$\quad S_{0,j} \leftarrow S_{0,j-1} + w((0,j-1),(0,j))$

$for \;\; i \leftarrow 1 \;\; to \;\; m$

$\quad for \;\; j \leftarrow 1 \;\; to \;\; n$

$\quad\quad S_{i,j} \leftarrow \max\left(S_{i-1,j} + w((i-1,j),(i,j)),\; S_{i,j-1} + w((i,j-1),(i,j)) \right)$

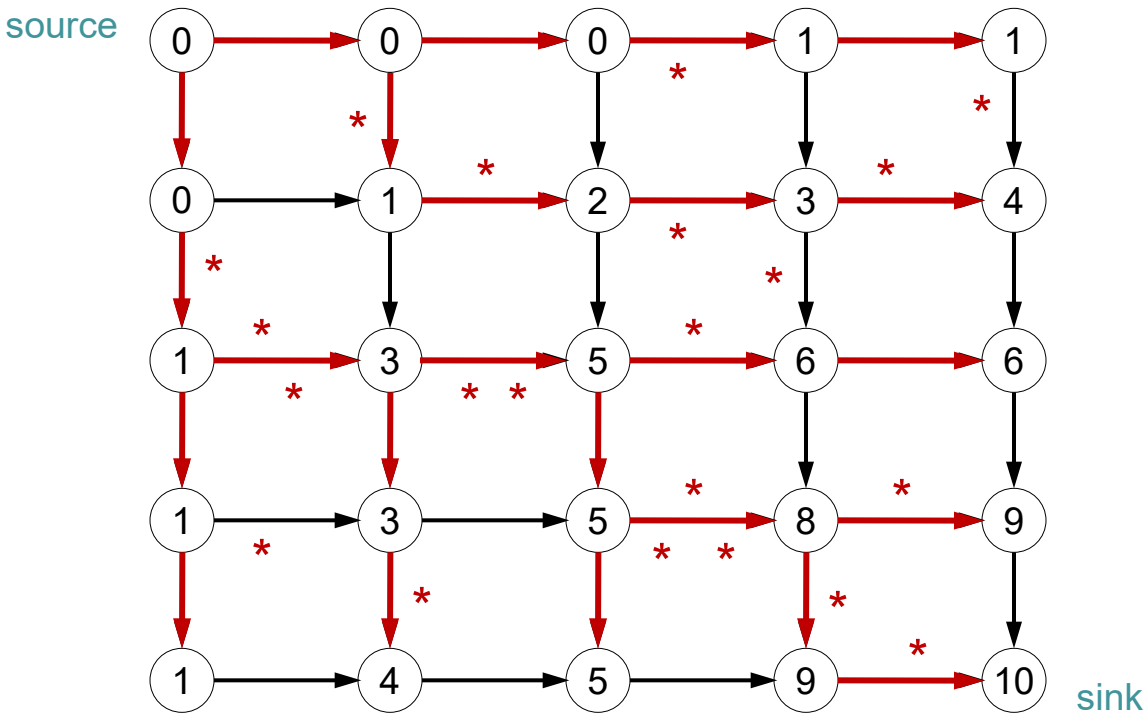
$return \;\; S_{m,n}$

Recursive Formula

$$S_{i,j} = \max\left(S_{i-1,j} + w\big((i-1,j),(i,j)\big),\; S_{i,j-1} + w\big((i,j-1),(i,j)\big) \right)$$

Example of Dynamic Programming

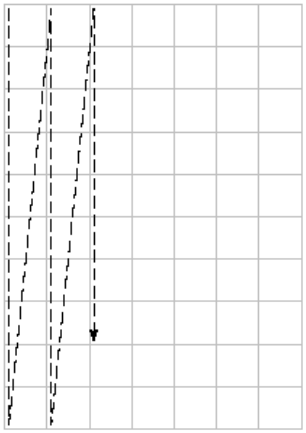
- **Example**

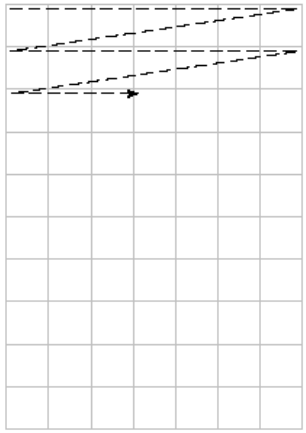


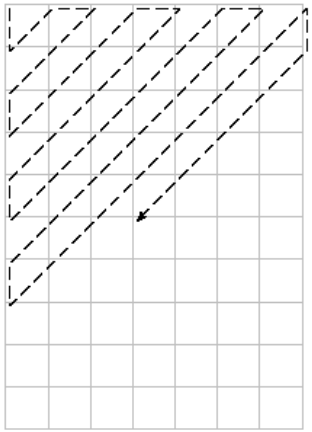
- **Runtime ?**

Traversing Strategies for Dynamic Programming

- **Three Different Strategies**
  - Column by column
  - Row by row
  - Along diagonals







Overview

1. Backgrounds
2. Manhattan Tourist Problem
3. Longest Common Subsequence Problem
4. Edit Distance
5. Global Sequence Alignment
6. Local Sequence Alignment
7. Alignment with Gap Penalty
8. Sequence Homolog Search
9. Multiple Sequence Alignment

Longest Common Subsequences (LCS)

- **Subsequence of x**
  - An ordered sequence of letters from x
  - Not necessarily consecutive
  - e.g., x="ATTGCTA", "AGCA" ?, "TCG" ?, "ATCT" ?, "TGAT" ?

- **Common Subsequence of x and y**
  - e.g., x="ATCTGAT" and y="TGCATA", "TCTA" ?, "TGAT" ?, "TATA" ?

- **Longest Common Subsequence (LCS) of x and y ?**

LCS in 2-Row Representation

- **Example**
  - x="ATCTGATG" (m=8), y="TGCATAC" (n=7)

	1	2		3		4	5	6	7	8	
x	A	T	—	C	—	T	G	A	T	G	—
y	—	T	G	C	A	T	—	A	—	—	C
		1	2	3	4	5		6			7

  - Matching position in x: 2 < 3 < 4 < 6
  - Matching Position in y: 1 < 3 < 5 < 6
  - Common subsequence: "TCTA"

LCS in 2-D Grid Representation

- **Edit Graph**
  - 2-D grid structure having diagonals on the position of the same letter

- **Example**
  - x="ATGTTAT" (m=7)
  - y="ATCGTAC" (n=7)
  - Strongest path in edit graph

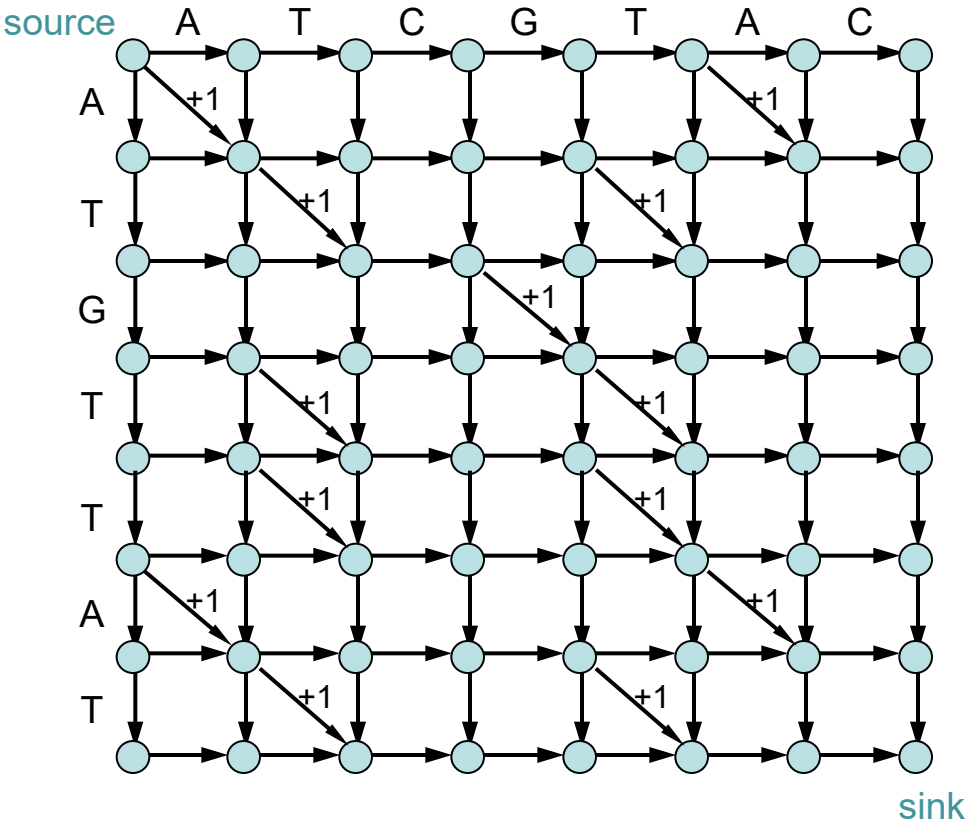
(0,0) → (1,1) → (2,2) →

(2,3) → (3,4) → (4,5) →

(5,5) → (6,6) → (7,6) →

(7,7)

		0	1	2	2	3	4	5	6	7	7
v	=		A	T	-	G	T	T	A	T	-
			|	|		|	|		|		
w	=		A	T	C	G	T	-	A	-	C
		0	1	2	3	4	5	5	6	6	7



Formulation of LCS Problem

- **Goal**
  - Finding the longest common subsequence (LCS) of two sequences (length-m, length-n)
  - Finding the strongest path from a source to a sink in a weighted edit graph
  - The path strength is measured by summing the weights on the path

- **Input**
  - A weighted edit graph G with source (0,0) and sink (m,n)

- **Output**
  - A strongest path in G from the source to the sink

Solving by Exhaustive Search

- **Algorithm**
  1) Enumerate all possible paths from the source to the sink
  2) Compute the path strength for all possible paths
  3) Find the strongest path

- **Problems ?**

Solving by Greedy Algorithm

- **Algorithm**
  1) Start from the source
  2) Select the edge having the highest weight
     (i.e., if there is a diagonal edge, select it. Otherwise, select one of the other edges.)
  3) Repeat (2) until it reaches the sink

- **Problems ?**

Solving by Dynamic Programming

- **Recursive Formula**

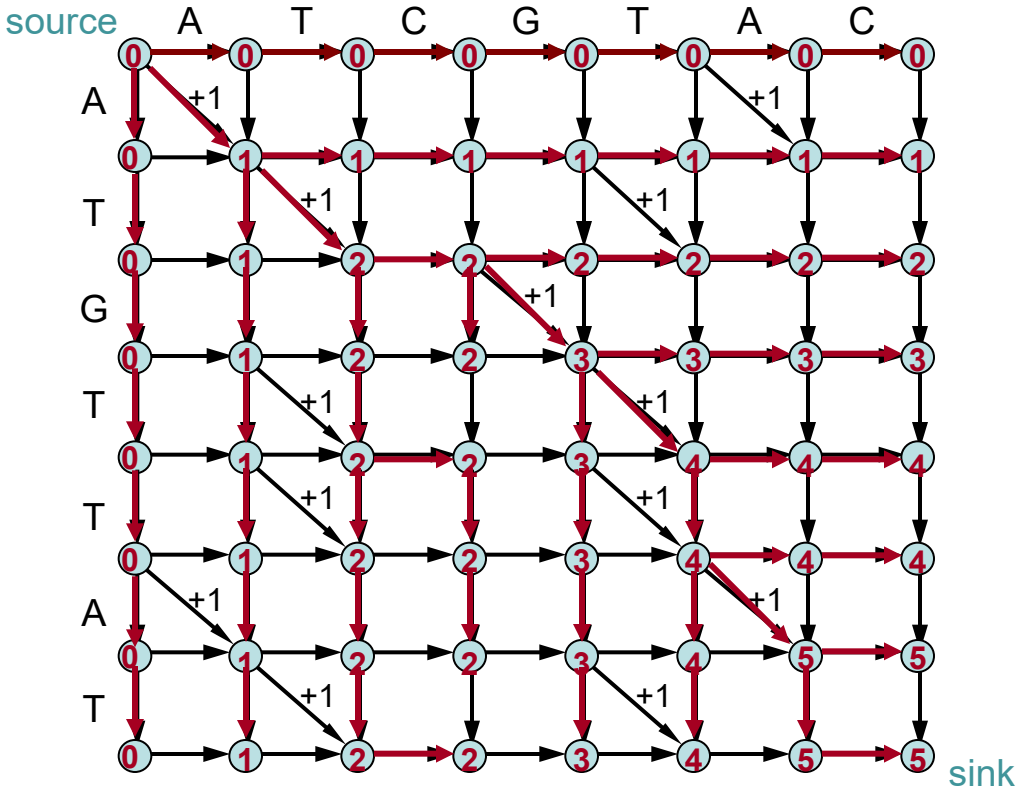
$$S_{i,j} = \max \begin{cases} S_{i-1,j} + 0 \\ S_{i,j-1} + 0 \\ S_{i-1,j-1} + 1 \quad if \;\; x_i = y_j \end{cases}$$

- **Algorithm**

```
LCS(x, y)
    for i ← 0 to m
        S_{i,0} ← 0
    for j ← 1 to n
        S_{0,j} ← 0
    for i ← 1 to m
        for j ← 1 to n
            if x_i = y_j
                S_{i,j} ← max(S_{i-1,i}, S_{i,j-1}, S_{i-1,j-1} + 1)
            else
                S_{i,j} ← max(S_{i-1,i}, S_{i,j-1})
    return S_{m,n}
```

Example of LCS

- **Example**
  - x="ATGTTAT" (m=7), y="ATCGTAC" (n=7)



Finding LCS

- **Storing Directions**

$$D_{i,j} \leftarrow \begin{cases} \text{“}\downarrow\text{”} & if \;\; S_{i,j} = S_{i-1,j} \\ \text{“}\rightarrow\text{”} & if \;\; S_{i,j} = S_{i,j-1} \\ \text{“}\searrow\text{”} & if \;\; S_{i,j} = S_{i-1,j-1} + 1 \end{cases}$$

- **Algorithm**

```
BACKTRACKING(D, x, i, j)
  if i > 0 and j > 0
    if D_{i,j} = "↓"
      BACKTRACKING(D, x, i − 1, j)
    else if D_{i,j} = "→"
      BACKTRACKING(D, x, i, j − 1)
    else
      BACKTRACKING(D, x, i − 1, j − 1)
      print x_i
```

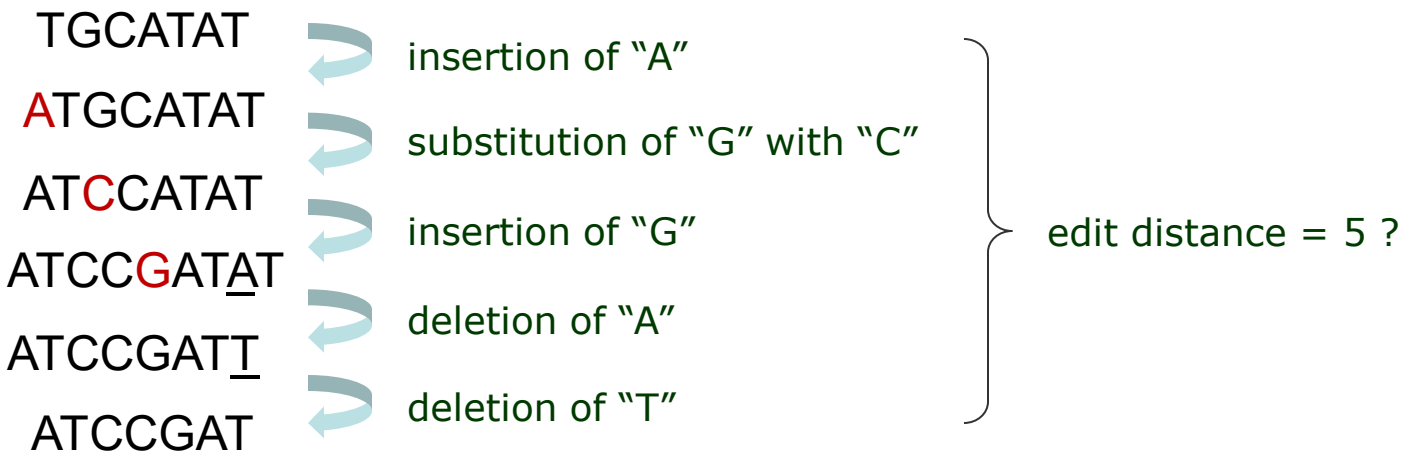
Overview

1. Backgrounds
2. Manhattan Tourist Problem
3. Longest Common Subsequence Problem
4. Edit Distance
5. Global Sequence Alignment
6. Local Sequence Alignment
7. Alignment with Gap Penalty
8. Sequence Homolog Search
9. Multiple Sequence Alignment

Definition of Edit Distance

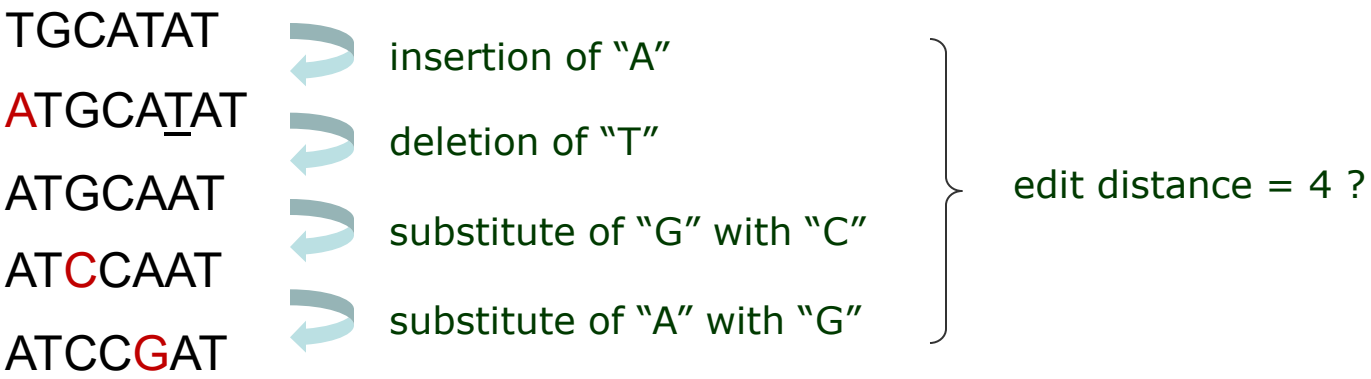
- **Definition**
  - Edit distance between two sequences x and y : the minimum number of editing operations (insertion, deletion, substitution) to transform x into y

- **Example**
  - x="TGCATAT" (m=7), y="ATCCGAT" (n=7)



Example of Edit Distance

- **Example**
  - x="TGCATAT" (m=7), y="ATCCGAT" (n=7)

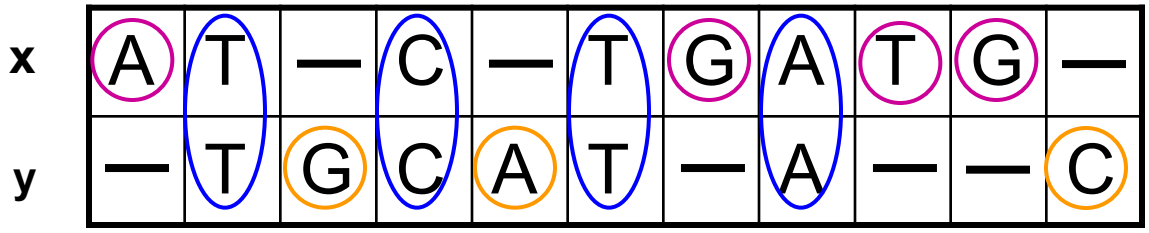


  - Can it be done in 3 steps?

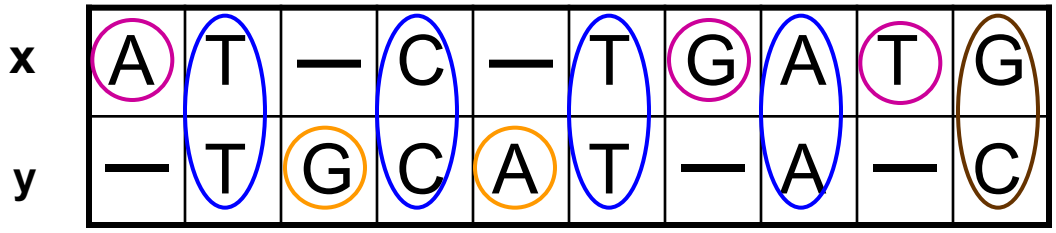
- **Features**
  - Allows comparison of two sequences of different lengths

Edit Distance in 2-Row Representation

- **Example**
  - x="ATCTGATG" (m=8), y="TGCATAC" (n=7)



4 matches
4 deletions
3 insertions



4 matches
3 deletions
2 insertions
1 substitution

Edit distance = **min** (#insertions + #deletions + #mismatches)

Edit Distance in 2D Grid Representation

- **Example**
  - x="ATCTGATG" (m=8), y="TGCATAC" (n=7)

Solving by Exhaustive Search or Greedy Algorithm

- **Exhaustive Search Algorithm**

- **Greedy Algorithm**

Solving by Dynamic Programming

- **Recursive Formula**

- **Dynamic Programming Algorithm**

  - Implementation result in 2D grid?

  - Runtime?

Overview

1. Backgrounds
2. Manhattan Tourist Problem
3. Longest Common Subsequence Problem
4. Edit Distance
5. Global Sequence Alignment
6. Local Sequence Alignment
7. Alignment with Gap Penalty
8. Sequence Homolog Search
9. Multiple Sequence Alignment

from LCS, Edit Distance to Sequence Alignment

- **LCS**
  - Allows only insertions and deletions - no substitutions
  - Scores 1 for a match and 0 for an insertion or deletion

- **Edit Distance**
  - Allows insertions, deletions, and substitutions
  - Scores 0 for a match and 1 for an insertion or deletion or substitution

- **Sequence Alignment**
  - Allows gaps (insertions and deletions) and mismatches (substitutions)
  - Uses any scoring schemes

Formulation of Global Sequence Alignment

- **Goal**
  - Finding the best alignment of two sequences under a given scoring schema
- **Input**
  - Two sequences x (length-m) and y (length-n), and a scoring schema
- **Output**
  - An alignment of x and y with the maximal score

Global Sequence Alignment with Basic Scoring Scheme

- **Basic Scoring Scheme**
  - Match premium: $+\alpha$
  - Mismatch penalty: $-\mu$
  - Insertion and deletion (gap) penalty: $-\sigma$

$$\text{Score} = \alpha\ \#\text{matches} - \mu\ \#\text{mismatches} - \sigma\ (\#\text{insertions} + \#\text{deletions})$$

- **Example in 2-D Grid**
  - x="ATGTTAT" (m=7), y="ATCGTAC" (n=7)

Solving by Exhaustive Search or Greedy Algorithm

- **Exhaustive Search Algorithm**

- **Greedy Algorithm**

Solving by Dynamic Programming

Recursive Formula

$$S_{i,j} = \max \begin{cases} S_{i-1,j} - \sigma \\ S_{i,j-1} - \sigma \\ S_{i-1,j-1} - \mu \quad if \;\; x_i \neq y_j \\ S_{i-1,j-1} + \alpha \quad if \;\; x_i = y_j \end{cases}$$

Dynamic Programming Algorithm

- Implementation result in 2D grid?
- Runtime?

GlobalAlignment(x, y)

$S_{0,0} \leftarrow 0$

$for \;\; i \leftarrow 1 \;\; to \;\; m$

$\quad S_{i,0} \leftarrow S_{i-1,0} - \sigma$

$for \;\; j \leftarrow 1 \;\; to \;\; n$

$\quad S_{0,j} \leftarrow S_{0,j-1} - \sigma$

$for \;\; i \leftarrow 1 \;\; to \;\; m$

$\quad for \;\; j \leftarrow 1 \;\; to \;\; n$

$\quad\quad if \;\; x_i = y_j$

$\quad\quad\quad S_{i,j} \leftarrow \max(S_{i-1,j} - \sigma, S_{i,j-1} - \sigma, S_{i-1,j-1} + \alpha)$

$\quad\quad else$

$\quad\quad\quad S_{i,j} \leftarrow \max(S_{i-1,j} - \sigma, S_{i,j-1} - \sigma, S_{i-1,j-1} - \mu)$

$return \;\; S_{m,n}$

Advanced Scoring Scheme

- **Advanced Scoring Scheme**
  - Varying scores for matches
  - Varying, strong penalties for mismatches
  - Relative likelihood of evolutionary relationship → Probability of mutations
  - Define scoring matrix for DNA or protein sequences

- **Scoring Matrix**
  - Also called substitution matrix
  - 4 × 4 array representation for DNA sequences
  - 20 × 20 array representation for protein sequences
  - Entry of $\delta(i,j)$ has the score between i and j

    → the rate at which i is substituted with j over time

Scoring Matrix Example

- **PAM (Point Accepted Mutations)**
  - Amino acid substitution frequency in mutations
  - PAM120 (results from 120 mutations per 100 residues) vs. PAM240
- **BLOSUM (Block Substitution Matrix)**
  - Substitution frequencies between clustered groups
  - BLOSUM-62 (results with a cut-off of 62% identity) vs. BLOSUM-50

(B)

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W
C	9																			
S	−1	3																		
T	−3	2	4																	
P	−3	1	−1	6																
A	−3	1	1	1	3															
G	−5	1	−1	−2	1	5														
N	−5	1	0	−2	0	0	4													
D	−7	0	−1	−2	0	0	2	5												
E	−7	−1	−2	−1	0	−1	1	3	5											
Q	−7	−2	−2	0	−1	−3	0	1	2	6										
H	−4	−2	−3	−1	−3	−4	2	0	−1	3	7									
R	−4	−1	−2	−1	−3	−4	−1	−3	−3	1	1	6								
K	−7	−1	−1	−2	−2	−3	1	−1	−1	0	−2	2	5							
M	−6	−2	−1	−3	−2	−4	−3	−4	−4	−1	−4	−1	0	8						
I	−3	−2	0	−3	−1	−4	−2	−3	−3	−3	−4	−2	−2	1	6					
L	−7	−4	−3	−3	−3	−5	−4	−5	−4	−2	−3	−4	−4	3	1	5				
V	−2	−2	0	−2	0	−2	−3	−3	−3	−3	−3	−3	−4	1	3	1	5			
F	−6	−3	−4	−5	−4	−5	−4	−7	−6	−6	−2	−4	−6	−1	0	0	−3	8		
Y	−1	−3	−3	−6	−4	−6	−2	−5	−4	−5	−1	−6	−6	−4	−2	−3	−3	4	8	
W	−8	−2	−6	−7	−7	−8	−5	−8	−8	−6	−5	1	−5	−7	−7	−5	−8	−1	−1	12

(A)

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

Solving by Dynamic Programming

- **Example in 2D Grid**
  - x="AGCATG" (m=6), y="ATGCGT" (n=6)

- **Recursive Formula**

$$S_{i,j} = \max \begin{cases} S_{i-1,j} + \delta(x_i, -) \\ S_{i,j-1} + \delta(-, y_j) \\ S_{i-1,j-1} + \delta(x_i, y_j) \end{cases}$$

- **Dynamic Programming Algorithm**
  - Implementation result in 2D grid?
  - Runtime?

Overview

1. Backgrounds
2. Manhattan Tourist Problem
3. Longest Common Subsequence Problem
4. Edit Distance
5. Global Sequence Alignment
6. Local Sequence Alignment
7. Alignment with Gap Penalty
8. Sequence Homolog Search
9. Multiple Sequence Alignment

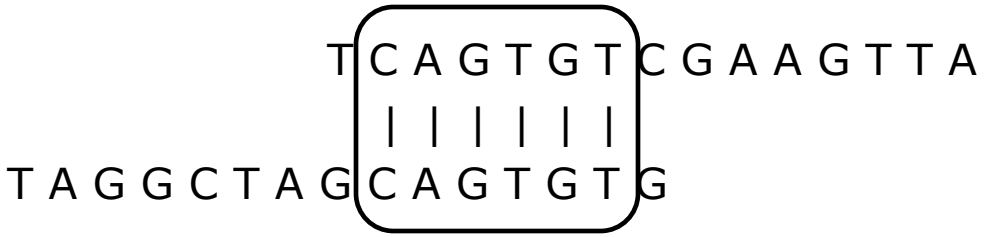
Local Sequence Alignment

- **Example**
  - x = “TCAGTGTCGAAGTTA”
  - y = “TAGGCTAGCAGTGTG”

- **Global Sequence Alignment**

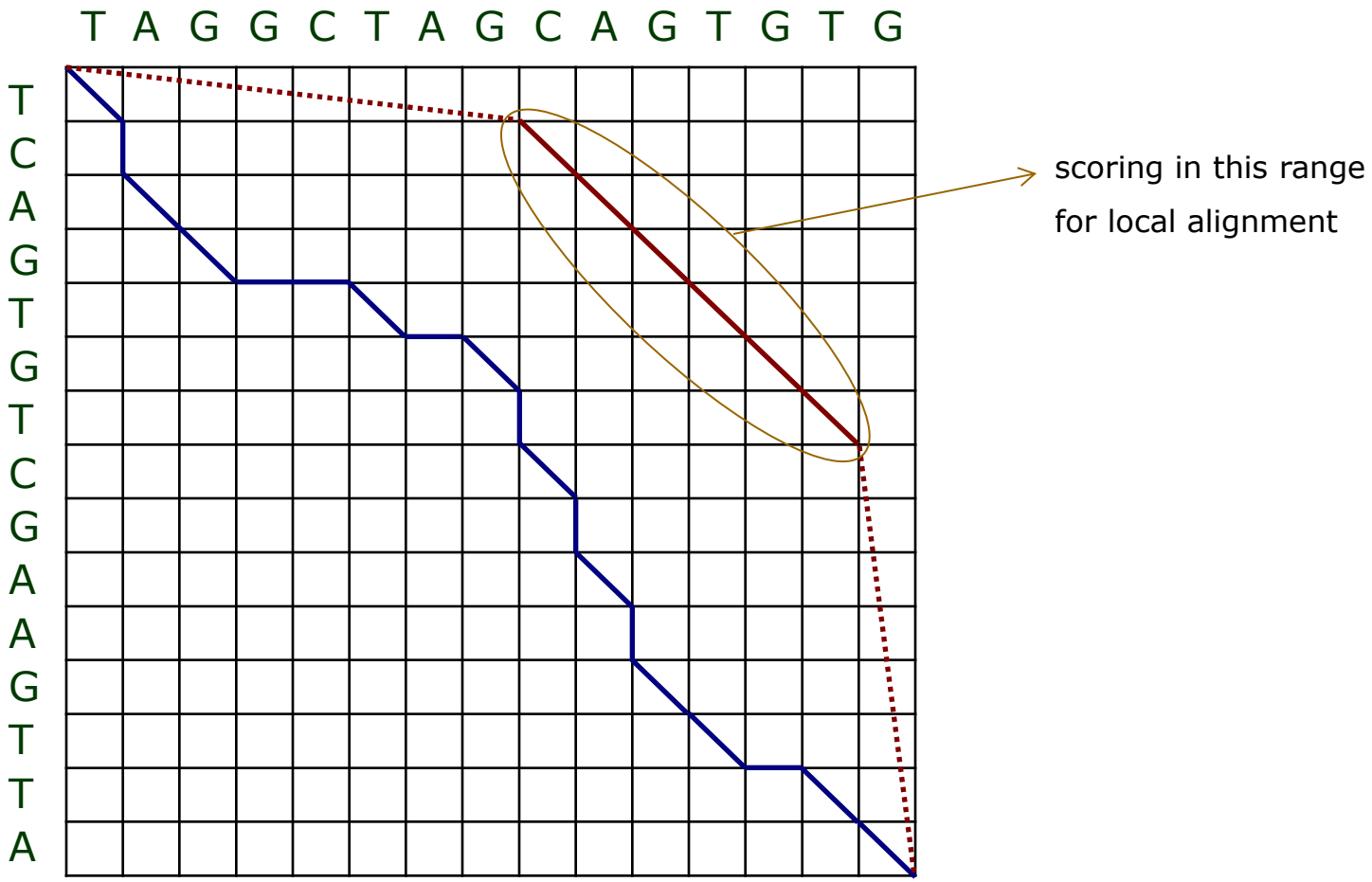
```
T C A G – – T – G T C G A A G T – T A
|   | |     |   |   |   |   | |   |
T – A G G C T A G – C – A – G T G T G
```

- **Local Sequence Alignment**



Local Sequence Alignment in 2-D Grid

- **Example**



Comparison of Global and Local Alignment

- **Global Alignment Problem**
  - Finds the path having the largest weight between vertices (0,0) and (m,n) in the edit graph

- **Local Alignment Problem**
  - Finds the path having the largest weight between two arbitrary vertices, (i, j) and (i', j'), in the edit graph

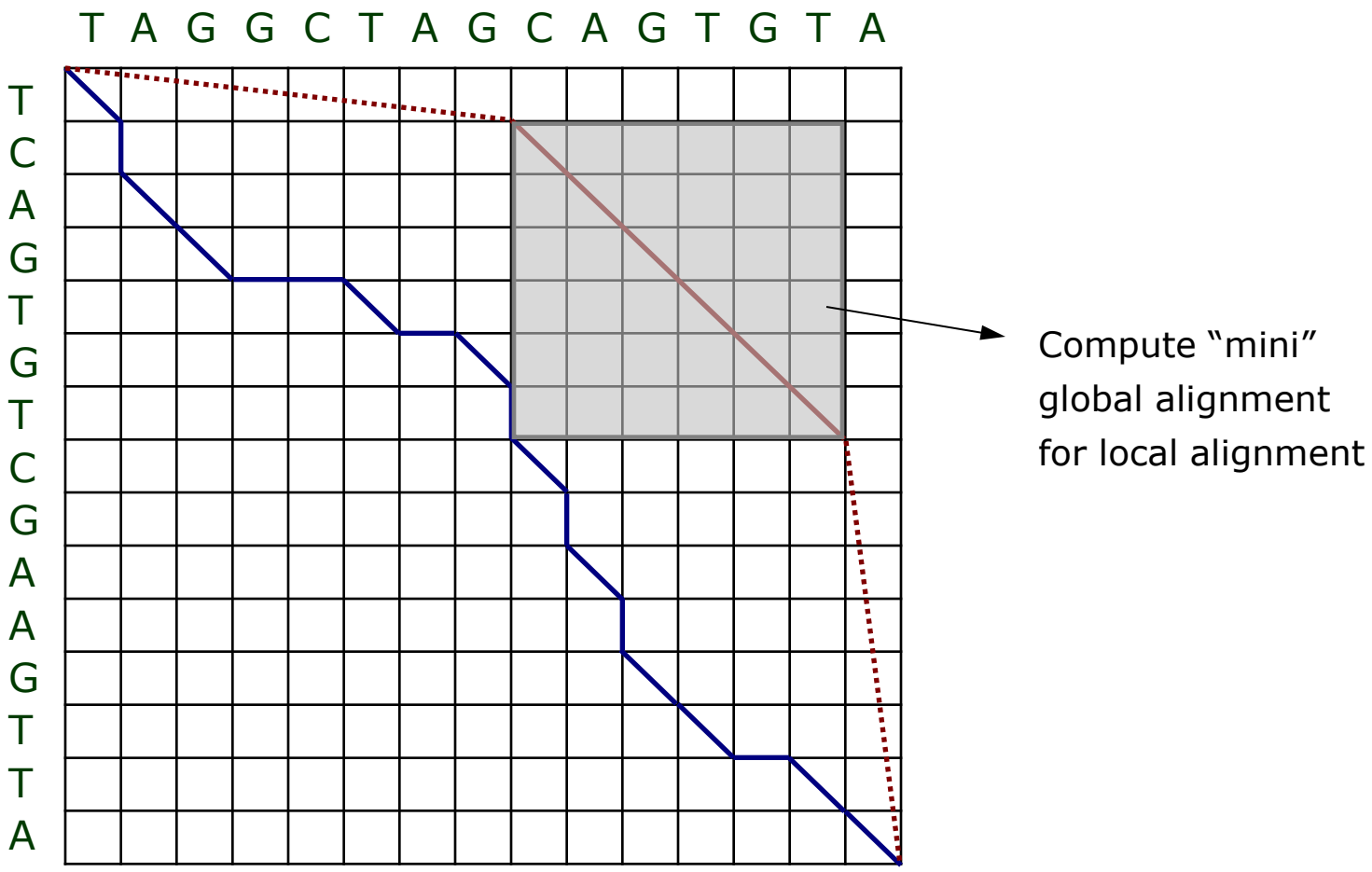
- **Score Comparison**
  - The score of local alignment must be greater than (or equal to) the score of global alignment

Formulation of Local Sequence Alignment

- **Goal**
  - Finding the best local alignment between two sequences
- **Input**
  - Two sequences x and y, and a scoring matrix δ
- **Output**
  - An alignment of substrings of x and y with the maximal score among all possible substrings of them

Implementation of Local Sequence Alignment

- **Strategy**



Solving by Exhaustive Search

- **Process**
  1) Enumeration of all possible pairs of substrings
  2) Global alignment for each pair of substrings

- **Process Re-written**
  1) Enumeration of all possible pairs of start position (i, j) and end position (i', j')
  2) Global alignment from each position (i, j) to each position (i', j')

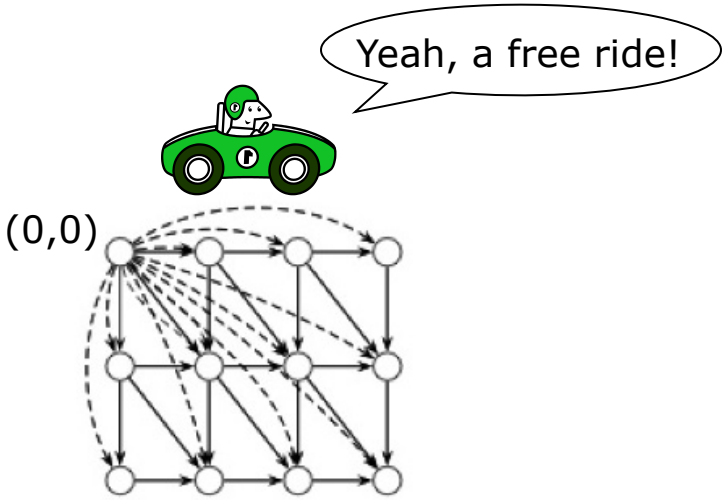
- **Runtime**
  - Suppose two sequences have the same length n
  - Global alignment :
  - Total runtime :

Solving by Exhaustive Search - Improved

- **Process Improved**
  1) Enumeration of all possible starting positions (i, j)
  2) Global alignment from each (i, j)

- **Runtime**
  - Suppose two sequences have the same length n
  - Global alignment :
  - Total runtime :

- **Solution**
  - Free ride !
  - Assigns 0 from (0,0) to any other nodes (i, j)



Solving by Dynamic Programming

- **Example in 2D Grid**
  - x="CGTCACT" (m=7), y="CTAGATC" (n=7)

- **Recursive Formula**

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

- **Dynamic Programming Algorithm**
  - Implementation result in 2D grid?
  - Runtime?

Overview

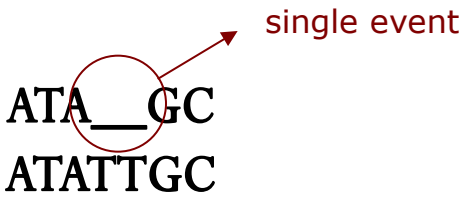
1. Backgrounds
2. Manhattan Tourist Problem
3. Longest Common Subsequence Problem
4. Edit Distance
5. Global Sequence Alignment
6. Local Sequence Alignment
7. Alignment with Gap Penalty
8. Sequence Homolog Search
9. Multiple Sequence Alignment

Scoring Insertions/Deletions

- **Naïve Approach**
  - -σ for 1 insertion/deletion,
  - -2σ for 2 consecutive insertions/deletions
  - -3σ for 3 consecutive insertions/deletions, etc.

    → too severe penalty for a series of 100 consecutive insertions/deletions

- **Example**
  - x="ATAGC", y="ATATTGC"



  - x="ATAGGC", y="ATGTGC"

ATAG_GC
AT_GTGC

Scoring Gaps of Insertions/Deletions

- **Gap**
  - Contiguous sequence of spaces in one of the rows
  - Contiguous sequence of insertions or deletions in 2-row representation

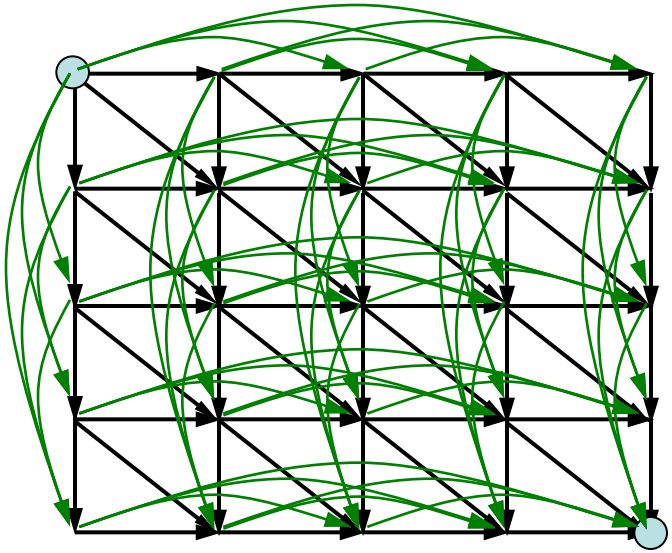
- **Linear Gap Penalty**
  - Score for a gap of length x : $-\sigma x$ (Naïve approach)

- **Constant Gap Penalty**
  - Score for a gap of length x : $-\rho$

- **Affine Gap Penalty**
  - Score for a gap of length x : $-(\rho + \sigma x)$
  - $-\rho$: gap opening (existence) penalty / $-\sigma$: gap extension penalty ($\rho \gg \sigma$)

Solving Constant/Affine Gap Penalty

- **Edit Graph Update**
  - Add “long” horizontal or vertical edges to the edit graph



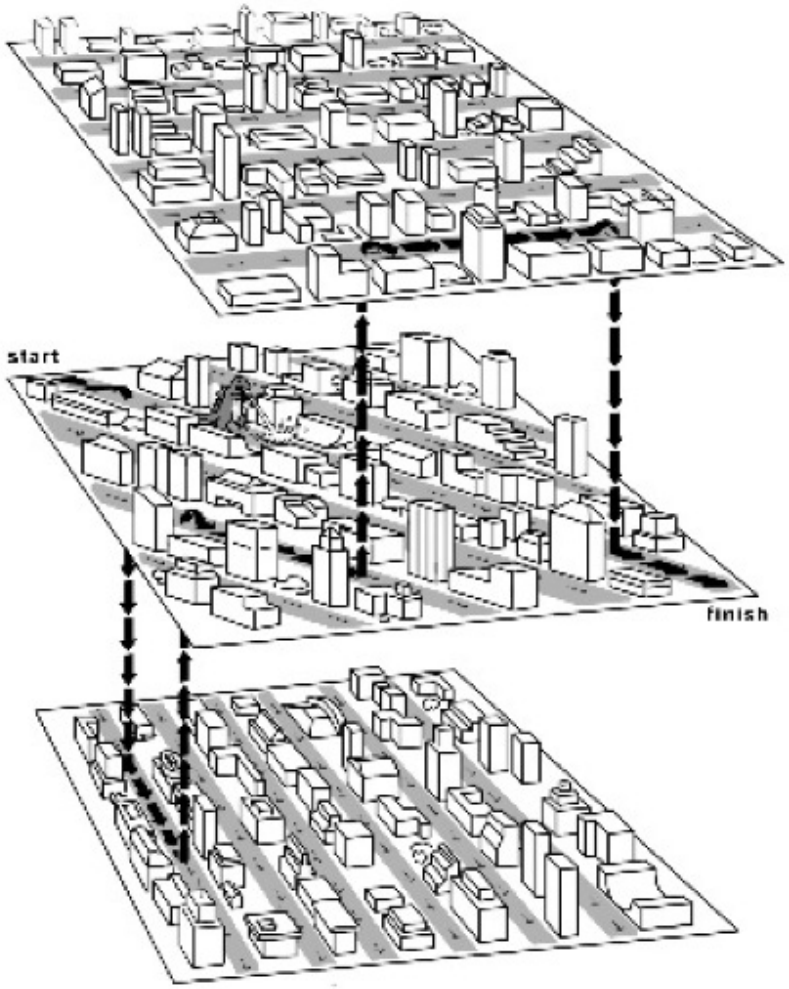
  - Runtime ?

Improved Solution for Constant/Affine Gap Penalty

- **3-Layer Grid Structure**
  - Middle layer (Main layer) for diagonal edges
    → Extends matches and mismatches
  - Upper layer for horizontal edges
    → Creates/extends gaps in a sequence y
  - Lower layer for vertical edges
    → Creates/extends gaps in a sequence x

- **Gap Opening / Gap Extension**
  - Gap opening penalty (-ρ) for jumping from middle layer to upper/lower layer
  - Gap extension penalty (-σ) for extending on upper/lower layer

Example of 3-Layer Grid



upper layer (gaps in *x*)

main layer (matches/mismatches)

lower layer (gaps in *y*)

Solving by Dynamic Programming

- **Example in 3-Layer 2D Grid**
  - x="GCATCTA" (m=7), y="CGTA" (n=4)

- **Recursive Formula**

$$S_{i,j} = \max \begin{cases} S_{i-1,j-1} + \delta(x_i, y_j) \\ S_{i,j}^{lower} \\ S_{i,j}^{upper} \end{cases}$$

→ match / mismatch

$$S_{i,j}^{lower} = \max \begin{cases} S_{i-1,j}^{lower} - \sigma \\ S_{i-1,j} - (\rho + \sigma) \end{cases}$$

→ continuing gap in *x*

→ starting gap in *x*

$$S_{i,j}^{upper} = \max \begin{cases} S_{i,j-1}^{upper} - \sigma \\ S_{i,j-1} - (\rho + \sigma) \end{cases}$$

→ continuing gap in *y*

→ starting gap in *y*

- **Dynamic Programming Algorithm**
  - Implementation result?
  - Runtime?

Overview

1. Backgrounds
2. Manhattan Tourist Problem
3. Longest Common Subsequence Problem
4. Edit Distance
5. Global Sequence Alignment
6. Local Sequence Alignment
7. Alignment with Gap Penalty
8. Sequence Homolog Search
9. Multiple Sequence Alignment

Sequence Homolog Search

- **Background**
  - Search similar sequences to a query sequence in a database
  - Computational issues
    - Dynamic programming are rigorous
    - But inefficient in searching a huge database
    - Need heuristic approaches

- **Sequence Homolog Searching Tools**
  - FASTA
  - BLAST

BLAST (1)

- **BLAST (Basic Local Alignment Search Tool)**
  - DNA / protein sequence alignment tool
  - Finds local alignments
  - Heuristic method in sequence search
  - Runs faster than FASTA

- **Algorithm**
  (1) Makes a list of words (word pairs) from the query sequence
  (2) Chooses high-scoring words
  (3) Searches database for matches (hits) with the high-scoring words
  (4) Extends the matches in both directions to find high-scoring segment pair (HSP)
  (5) Selects the sequence which has two or more HSPs for local alignment

BLAST (2)

- **DFA (Deterministic Finite Automata) Analysis**
  - Build DFA using high-scoring words
  - Read sequences in database and trace DFA
  - Output the positions for hits
- **BLAST package**

	query sequence	database
blastp	protein	protein
blastn	DNA	DNA
blastx	DNA (all reading frames)	protein
tblastn	protein	DNA (all reading frames)
tblastx	DNA (all reading frames)	DNA (all reading frames)

Homolog Search Results

BLAST Search Results

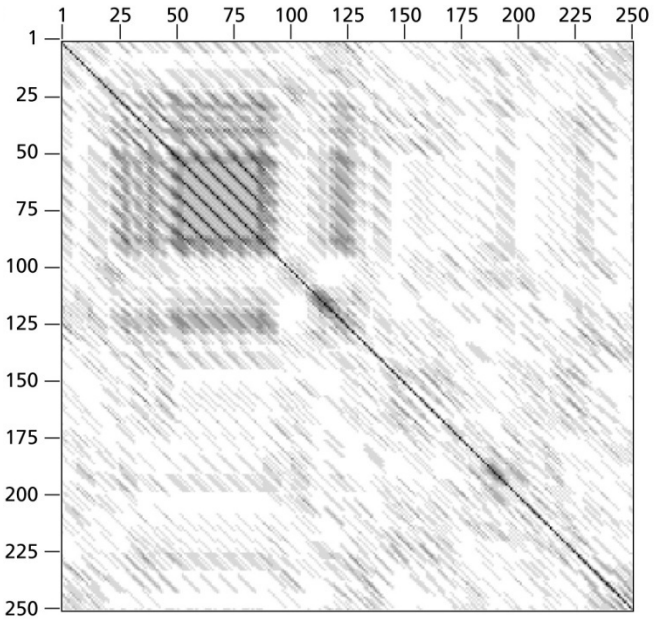
sp|P32871|P11A_BOVIN	PHOSPHATIDYLINOSITOL 3-KINASE CATALYTI...	680	0.0
sp|P42336|P11A_HUMAN	PHOSPHATIDYLINOSITOL 3-KINASE CATALYT...	676	0.0
sp|P42337|P11A_MOUSE	PHOSPHATIDYLINOSITOL 3-KINASE CATALYT...	674	0.0
sp|P42338|P11B_HUMAN	PHOSPHATIDYLINOSITOL 3-KINASE CATALYT...	338	9e-93
sp|O35904|P11D_MOUSE	PHOSPHATIDYLINOSITOL 3-KINASE CATALYT...	332	7e-91
sp|O00329|P11D_HUMAN	PHOSPHATIDYLINOSITOL 3-KINASE CATALYT...	331	2e-90
sp|P47473|RIR1_MYCGE	RIBONUCLEOSIDE-DIPHOSPHATE REDUCTASE A...	34	0.59

E-value

- Average number of alignments with a score of at least S that would be expected by chance alone in searching a database of n sequences
- Ranges of E-value: 0 ~ n
- High alignment score S → Low E-value
- Low alignment score S → High E-value
- Factors
  - Alignment score
  - The number of sequences in the database
  - Sequence length
- Default E-value threshold: 0.001 ~ 0.01

Filtering

- **Low-Complexity Region**
  - Highly biased amino acid composition
  - Lowers significant hits in sequence alignment
  - BLAST filters the query sequence for low-complexity regions and mark “X”




```
>sp[P04156]PRIO_HUMAN  MAJOR PRION PROTEIN PRECURSOR (PRP) (PRP27-30) (PRP33-35C)
           Length = 253

Score =  312 bits (792), Expect = 5e-85
Identities = 154/236 (65%), Positives = 154/236 (65%)

Query: 64   MANLGCWMLVLFVATWSDLGLCKKRPKPGGWNTGGSRYPGQGSPGGNRYXXXXXXXXXXX 123
            MANLGCWMLVLFVATWSDLGLCKKRPKPGGWNTGGSRYPGQGSPGGNRY
Sbjct: 1    MANLGCWMLVLFVATWSDLGLCKKRPKPGGWNTGGSRYPGQGSPGGNRYPPQGGGGWGQP 60

Query: 124  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXTHSQWNKPSKPKTNMKHMXXXXXXXX 183
            XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXTHSQWNKPSKPKTNMKHM
Sbjct: 1    HGGGWGQPHGGGWGQPHGGGWGQPHGGGWGQGGGTHSQWNKPSKPKTNMKHMAGAAAAGA 120

Query: 184  XXXXXXXXXXXXXXXRPIIHFGSDYEDRYYRENMHRYPNQVYYRPMDEYSNQNNFVHDCV 243
                          RPIIHFGSDYEDRYYRENMHRYPNQVYYRPMDEYSNQNNFVHDCV
Sbjct: 121  VVGGLGGYMLGSAMSRPIIHFGSDYEDRYYRENMHRYPNQVYYRPMDEYSNQNNFVHDCV 180

Query: 244  NITIKQHXXXXXXXXXXXXXXDVKMMERVVEQMCITQYERESQAYYQRGSSMVLFS 299
            NITIKQH               DVKMMERVVEQMCITQYERESQAYYQRGSSMVLFS
Sbjct: 181  NITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYERESQAYYQRGSSMVLFS 236
```

Overview

1. Backgrounds
2. Manhattan Tourist Problem
3. Longest Common Subsequence Problem
4. Edit Distance
5. Global Sequence Alignment
6. Local Sequence Alignment
7. Alignment with Gap Penalty
8. Sequence Homolog Search
9. Multiple Sequence Alignment

Pairwise Alignment vs. Multiple Alignment

- **Pairwise Sequence Alignment**
  - Aligning two sequences
  - Sometimes two sequences are functionally similar or have a common ancestor although they have weak sequence similarity

- **Multiple Sequence Alignment**
  - Aligning more than two sequences simultaneously
  - Finds invisible similarity in pairwise alignment

Alignment of 3 Sequences

- **Alignment of 2 Sequences**
  - Described in a 2-row representation
  - Best alignment is found in a 2-D grid by dynamic programming

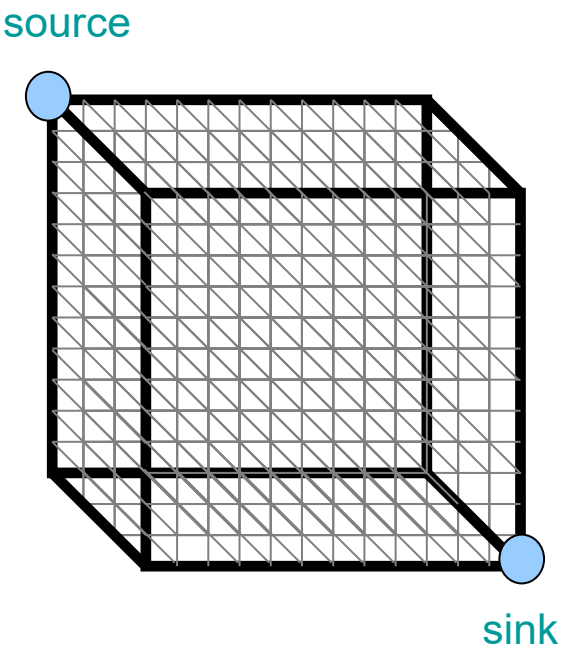
- **Alignment of 3 Sequences**
  - Described in a 3-row representation
  - Example: x="ATGTG", y="ACGTA", z="ATCTG"

x :		A	T	–	G	T	G	–
y :		A	–	C	G	T	–	A
z :		A	T	C	–	T	G	–

  - Best alignment is found in a 3-D grid by dynamic programming

Alignment in 3-D Grid

- **3-D Edit Graph**
  - 3-D grid structure (cube) with diagonals in each cell



- **Example**

	0	1	2	2	3	4	5	5
x :		A	T	–	G	T	G	–

	0	1	1	2	3	4	4	5
y :		A	–	C	G	T	–	A

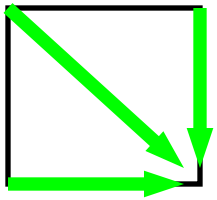
	0	1	2	3	3	4	5	5
z :		A	T	C	–	T	G	–

  - Path in 3-D grid :

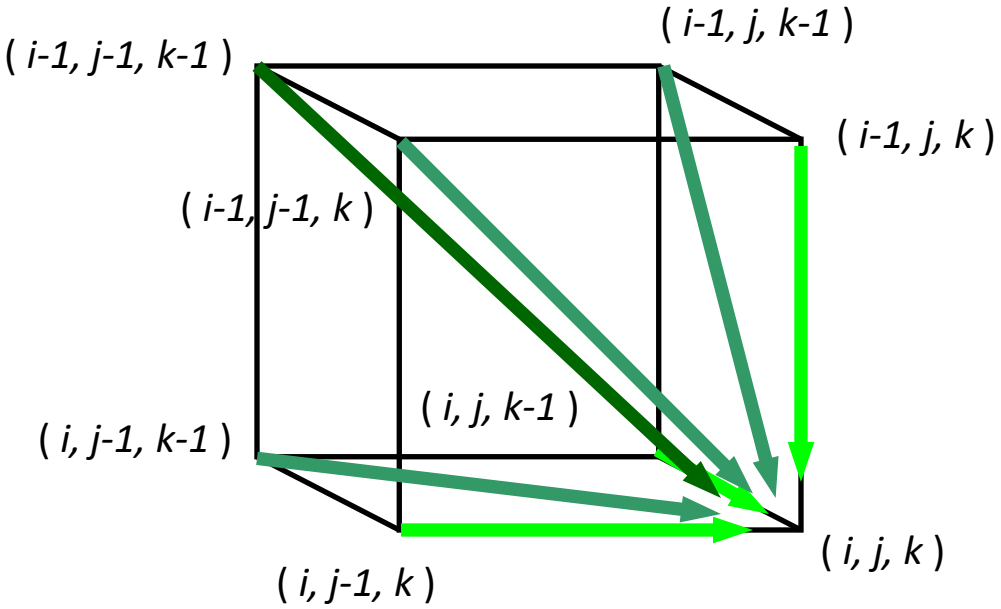
(0,0,0) → (1,1,1) → (2,1,2) → (2,2,3) → (3,3,3) → (4,4,4) → (5,4,5) → (5,5,5)

3-D Grid Unit

- **2-D Grid Unit Cell**
  - Maximum 3 edges in each unit of 2-D grid



- **3-D Grid Unit Cell**
  - Maximum 7 edges in each unit of 3-D grid



Solving by Dynamic Programming

- **Formula**

$$S_{i,j,k} = \max \begin{cases} S_{i-1,j,k} + \delta(x_i, -, -) \\ S_{i,j-1,k} + \delta(-, y_i, -) \\ S_{i,j,k-1} + \delta(-, -, z_k) \\ S_{i-1,j-1,k} + \delta(x_i, y_j, -) \\ S_{i-1,j,k-1} + \delta(x_i, -, z_k) \\ S_{i,j-1,k-1} + \delta(-, y_j, z_k) \\ S_{i-1,j-1,k-1} + \delta(x_i, y_j, z_k) \end{cases}$$

  - δ (x, y, z) is the entry of 3-D scoring matrix

- **Runtime ?**

from 3-D Alignment to Multiple Alignment

- **Alignment of k Sequences**
  - Able to be solved by dynamic programming in k-D grid
  - Runtime ?

- **Conclusion**
  - Dynamic programming for pairwise alignment can be extended to multiple alignment
  - However, computationally impractical
  - How can we solve this problem ?

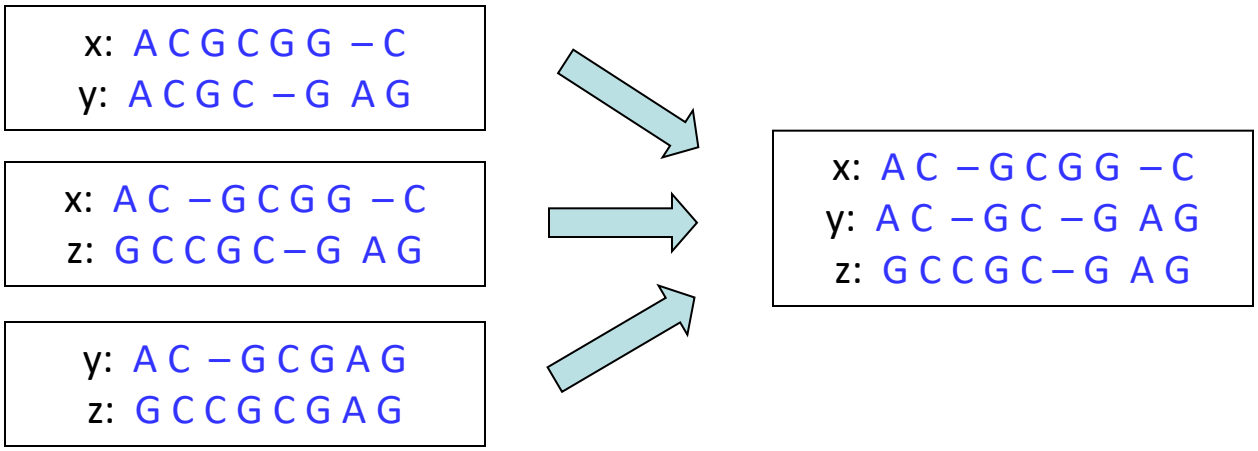
    → We need heuristic algorithms!!

Heuristics of Multiple Alignment

- **Background**
  - Implementing pairwise alignment (2-D alignment) k times is better than implementing k-D multiple alignment once

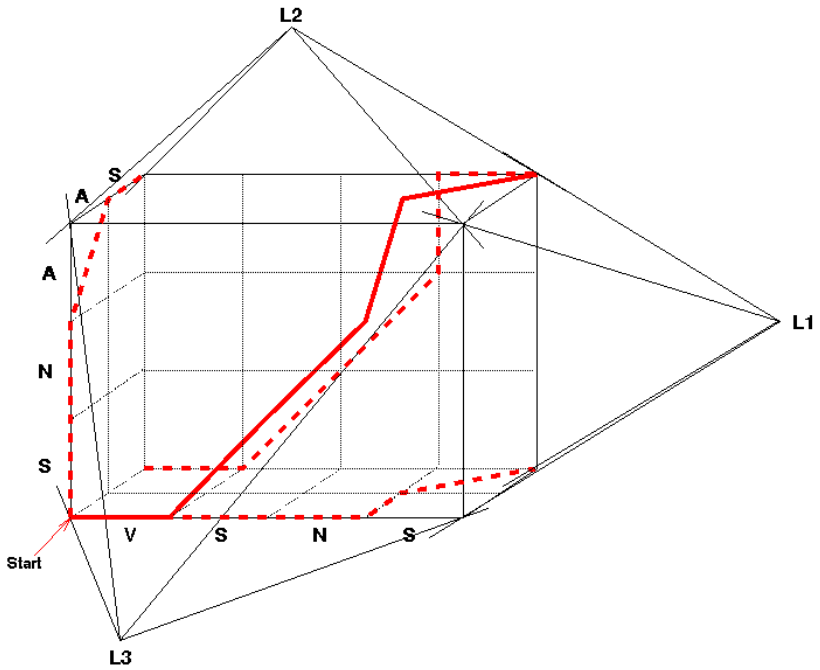
- **Heuristic Process**
  1) Implementing all possible pairwise alignments
  2) Combining the most similar pair iteratively

- **Why it is heuristic?**
  - Can we construct a multiple alignment that induces pairwise alignments ?



Projection of Alignments

- **Projection**



- **Conclusion**
  - Can’t infer optimal multiple alignment from all optimal pairwise alignments
  - Example?

Solving by Greedy Algorithm

- **Process**
  1) Choose the most similar pair of sequences
  2) Merge them into a new sequence
  3) Choose the most similar sequence to the new sequence
  4) Repeat (2) and (3) until choosing all sequences

- **Example**
  - Step 1

s1: GATTCA
s2: GTCTGA
s3: GATATT
s4: GTCAGC

s2 **GTC**T**G**A
s4 **GTC**A**G**C

s1 **GAT**T**CA**--
s4 **G**-**T**-**CA**GC

s1 **GAT**-**TCA**
s2 **G**-**TCTGA**

s2 **G**-**TCTGA**
s3 **GATAT**-T

s1 **GAT**-**TCA**
s3 **GATAT**-T

s3 **GAT**-**ATT**
s4 **G**-**TCAGC**

Solving by Greedy Algorithm – Cont'

- **Example - continued**
  - Step 2

    s2 **GTC**T**G**A
    s4 **GTC**A**G**C

    $s_{2,4}$ **GTC**t/a**G**a/c
    (called *profile* or *consensus sequence*)
  - Step 3

    s_1 GATTCA
    s_3 GATATT
    $s_{2,4}$ GTCt/aGa/c

- **Features**
  - k-way alignment (alignment of k sequences) → Runtime ?
  - Greedy algorithm

    → Not optimal multiple sequence alignment

Progressive Alignment

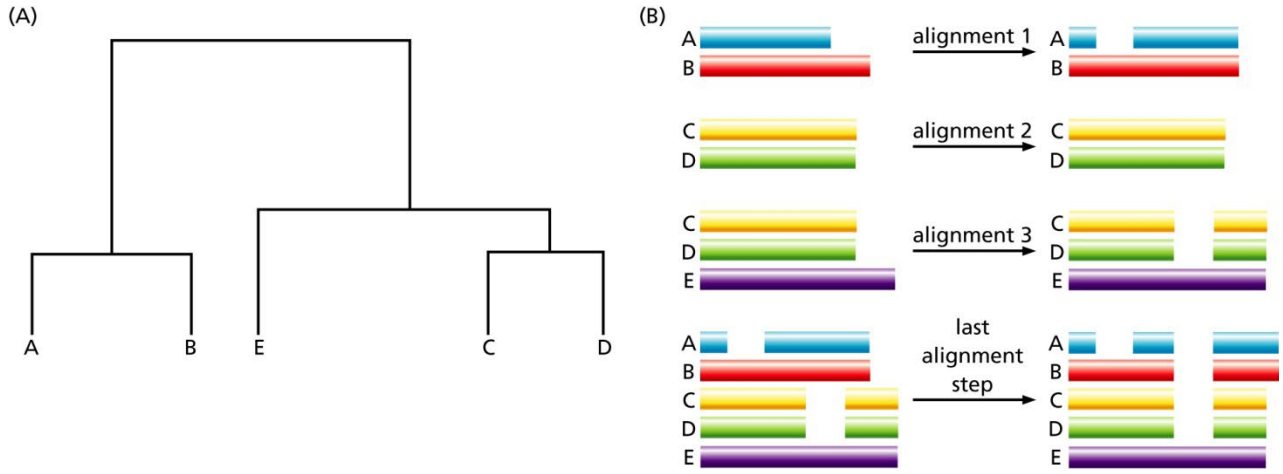
- **Features**
  - A variation of greedy algorithm (more intelligent strategy on each step)
  - Also called hierarchical method
  - Uses profiles to compare sequences
  - Gaps are permanent (“once a gap, always a gap”)
  - Works well for close sequences

- **Process**
  - Stage 1
    - Computes sequence identity of all possible pairs of sequences (identity = #match / sequence length)
    - Makes a similarity matrix

	v_1	v_2	v_3	v_4	…
v_1	–				
v_2	.17	–			
v_3	.87	.28	–		
v_4	.59	.33	.62	–	

Progressive Alignment – Cont'

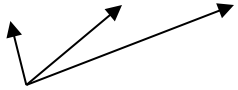
- **Process**
  - Stage 2
    - Creates a guide tree using the similarity matrix
  - Stage 3
    - Applies a series of pairwise alignment following the guide tree



Application of Progressive Alignment

- **ClustalW / ClustalX**
  - Popular multiple alignment tool
  - Adopts the progressive multiple alignment

```
FOS_RAT          PEEMSVTS-LDLTGGLPEATTPESEEAFTLPLLNDPEPK-PSLEPVKNISNMELKAEPFD
FOS_MOUSE        PEEMSVAS-LDLTGGLPEASTPESEEAFTLPLLNDPEPK-PSLEPVKSISNVELKAEPFD
FOS_CHICK        SEELAAATALDLG----APSPAAAEEAFALPLMTEAPPAVPPKEPSG--SGLELKAEPFD
FOSB_MOUSE       PGPGPLAEVRDLPG-----STSAKEDGFGWLLPPPPPPP-----------------LPFQ
FOSB_HUMAN       PGPGPLAEVRDLPG-----SAPAKEDGFSWLLPPPPPPP-----------------LPFQ
                 .   . :   ** .     :..  *:.*   *   . *                   **:
```



Dots and stars show how well-conserved a column is

Scoring Schemes for Multiple Sequence Alignment

- **Number of Matches**
  - Multiple longest common subsequence score
  - A column is a “match” if all the letters in the column are the same

    AAA …
    AAG …
    AAT …
    ATC …

  - Only good for very similar sequences

- **Sum-of-Pair Scoring**

- **Entropy-Based Scoring**

Sum-of-Pair Scoring

- **Sum-of-Pairs Scoring in Multiple Alignment**
  - Consider pairwise alignments imposed by a multiple alignment of k sequences
  - Denote the score of the pairwise alignment between a_i and a_j as $S^*(a_i, a_j)$
  - Sum up the pairwise scores for a multiple alignment:

$$S(a_1, a_2, \cdots, a_k) = \sum_{i,j} S*(a_i, a_j)$$

- **Example**
  - Aligning 4 sequences, a_1, a_2, a_3, and a_4, by

$$S(a_1, a_2, a_3, a_4) = S*(a_1, a_2) + S*(a_1, a_3) + S*(a_1, a_4) + S*(a_2, a_3) + S*(a_2, a_4) + S*(a_3, a_4)$$

Entropy-Based Scoring

- **Entropy in Information Theory**
  - A measure of the uncertainty associated with a random variable

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i)$$

- **Entropy-Based Scoring in Multiple Alignment**
  1) Define frequencies for the occurrence of each letter on each column
  2) Compute entropy of each column
  3) Sum all entropies over all columns

- **Example**

AAA
AAG
AAT
ATC

Questions?

- Lecture Slides on the Course Website, “https://ads.yonsei.ac.kr/faculty/biocomputing”