CSI 5V93

Sequence Alignment

Young-Rae Cho

Associate Professor

Department of Computer Science

Baylor University

BAYLOR

Sequence Alignment

- Definition
  - Arranging two or more sequences by inserting gaps to maximize their similarity score
- Usage
  - To measure similarity between two sequences
  - To identify regions of high similarity between two sequences
- Applications in Bioinformatics
  - Given gene sequences, infer their evolutionary distance
  - Given gene sequences of known functions, predict the functions of newly sequenced genes

BAYLOR

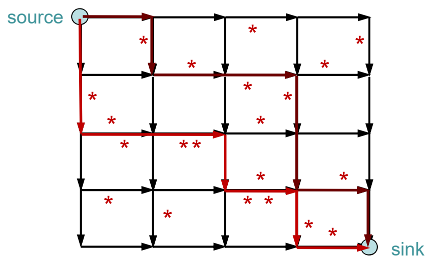
Overview

- ➢ *Manhattan Tourist Problem*
- ➢ Longest Common Subsequence Problem
- ➢ Edit Distance
- ➢ Global Sequence Alignment
- ➢ Local Sequence Alignment
- ➢ Alignment with Gap Penalty
- ➢ Multiple Sequence Alignment

BAYLOR

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



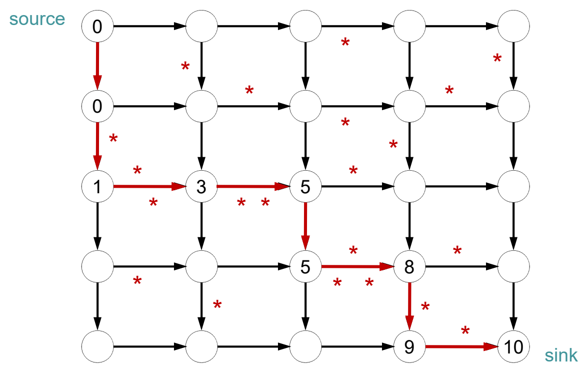
BAYLOR

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 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
  - (3) Repeat (2) until it reaches the *sink*
- Problems ?
- Runtime ?

Solving by Recursive Algorithm

- Algorithm

```
$\text{MTP}(m, n)$
    $if\ \ m = 0\ \ and\ \ n = 0$
        $return\ \ 0$
    $else\ \ if\ \ m = 0\ \ and\ \ n \neq 0$
        $return\ \ \text{MTP}(m, n-1) + w((m, n-1), (m, n))$
    $else\ \ if\ \ m \neq 0\ \ and\ \ n = 0$
        $return\ \ \text{MTP}(m-1, n) + w((m-1, n), (m, n))$
    $else$
        $x \leftarrow \text{MTP}(m-1, n) + w((m-1, n), (m, n))$
        $y \leftarrow \text{MTP}(m, n-1) + w((m, n-1), (m, n))$
        $return\ \ max(x, y)$
```

- Problems ?
- Runtime ?

BAYLOR

Solving by Dynamic Programming

- Recursive Formula

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

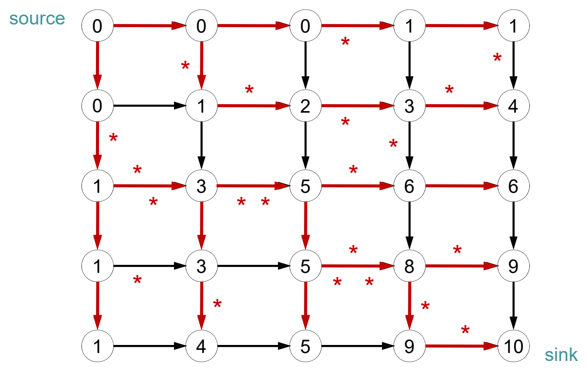
- Algorithm

```
$\text{MTP}(m, n)$
    $S_{0,0} \leftarrow 0$
    $for\ \ i \leftarrow 1\ \ to\ \ m$
        $S_{i,0} \leftarrow S_{i-1,0} + w((i-1, 0), (i, 0))$
    $for\ \ j \leftarrow 1\ \ to\ \ n$
        $S_{0,j} \leftarrow S_{0,j-1} + w((0, j-1), (0, j))$
    $for\ \ i \leftarrow 1\ \ to\ \ m$
        $for\ \ j \leftarrow 1\ \ to\ \ n$
            $S_{i,j} \leftarrow \max\left(\ S_{i-1,j} + w((i-1, j), (i, j)),\ S_{i,j-1} + w((i, j-1), (i, j))\ \right)$
    $return\ \ S_{m,n}$
```

BAYLOR

Example of Dynamic Programming

- Example

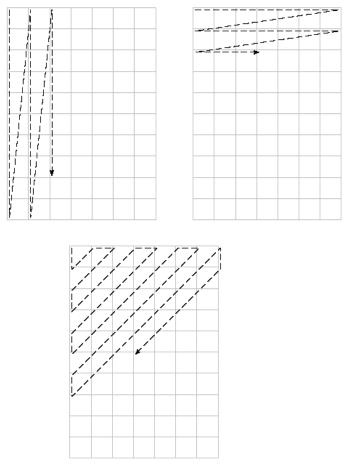


- Runtime ?

BAYLOR

Traversing Strategies

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



BAYLOR

Overview

- Manhattan Tourist Problem
- *Longest Common Subsequence Problem*
- Edit Distance
- Global Sequence Alignment
- Local Sequence Alignment
- Alignment with Gap Penalty
- Multiple Sequence Alignment

BAYLOR

Longest Common Subsequences (1)

- 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 ?

BAYLOR

Longest Common Subsequences (2)

- Definition of LCS
  - Given two sequences, $v = \langle v_1 v_2 \dots v_m \rangle$ and $w = \langle w_1 w_2 \dots w_n \rangle$, LCS of v and w is a sequence of positions in

    v: $1 \le i_1 < i_2 < \dots < i_t \le m$

    and a sequence of positions in

    w: $1 \le j_1 < j_2 < \dots < j_t \le n$

    such that i_t-th letter of v equals to j_t-letter of w, and t is maximal

LCS in 2-Row Representation (1)

- 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

  - Position in x: $2 < 3 < 4 < 6$
  - Position in y: $1 < 3 < 5 < 6$
  - LCS: "TCTA"

LCS in 2-Row Representation (2)

- Example - Continued
  - 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

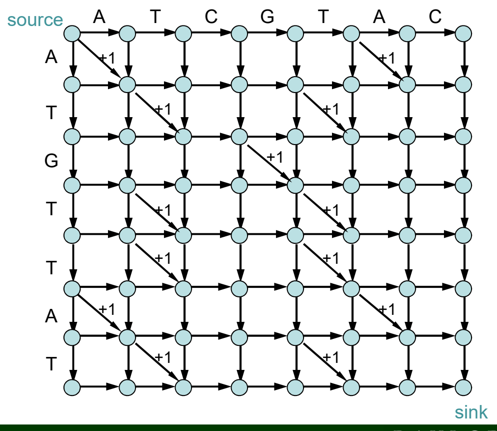
	1	2	3	4	5		6	7	8		
x	A	T	C	T	G	—	A	T	G	—	—
y	—	T	—	—	G	C	A	T	—	A	C
		1			2	3	4	5		6	7

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*

BAYLOR

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 ?

BAYLOR

Solving by (Iterative) 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 ?

- Runtime ?

Solving by Dynamic Programming

- Recursive Formula

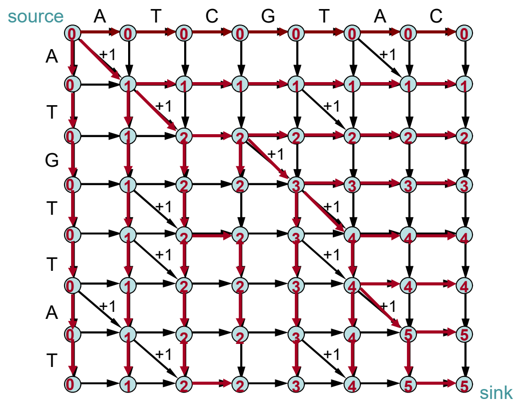
$$S_{i,j} = \max \begin{cases} S_{i-1,j} + 0 \\ S_{i,j-1} + 0 \\ S_{i-1,j-1} + 1 & if \;\; x_i = y_j \end{cases}$$

- Algorithm

```
LCS$(x, y)$
  $for \;\; i \leftarrow 0 \;\; to \;\; m$
    $S_{i,0} \leftarrow 0$
  $for \;\; j \leftarrow 1 \;\; to \;\; n$
    $S_{0,j} \leftarrow 0$
  $for \;\; i \leftarrow 1 \;\; to \;\; m$
    $for \;\; j \leftarrow 1 \;\; to \;\; n$
      $if \;\; x_i = y_j$
        $S_{i,j} \leftarrow \max\left(S_{i-1,i},\; S_{i,j-1},\; S_{i-1,j-1} + 1\right)$
      $else$
        $S_{i,j} \leftarrow \max\left(S_{i-1,i},\; S_{i,j-1}\right)$
  $return \;\; S_{m,n}$
```

Example of LCS

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



Finding LCS

- Storing Directions

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

- Backtracking

```
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

- Manhattan Tourist Problem
- Longest Common Subsequence Problem
- *Edit Distance*
- Global Sequence Alignment
- Local Sequence Alignment
- Alignment with Gap Penalty
- Multiple Sequence Alignment

BAYLOR

Edit Distance (1)

- 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)

TGCATAT
insertion of "A"
ATGCATAT
substitution of "G" with "C"
ATCCATAT
insertion of "G"
ATCCGATAT
deletion of "A"
ATCCGATT
deletion of "T"
ATCCGAT

edit distance = 5 ?

BAYLOR

Edit Distance (2)

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

TGCATAT	
	insertion of "A"
ATGCATAT	
	deletion of "T"
ATGCAAT	
	substitute of "G" with "C"
ATCCAAT	
	substitute of "A" with "G"
ATCCGAT	

edit distance = 4 ?

  - Can it be done in 3 steps?

- Features
  - Allows comparison of two sequences of different lengths

Edit Distance in 2-Row Representation

- Example in 2-row representation
  - x="ATCTGATG" (m=8), y="TGCATAC" (n=7)

x	A	T	—	C	—	T	G	A	T	G	—
y	—	T	G	C	A	T	—	A	—	—	C

4 matches
4 deletions
3 insertions

x	A	T	—	C	—	T	G	A	T	G
y	—	T	G	C	A	T	—	A	—	C

4 matches
3 deletions
2 insertions
1 substitution

Edit distance = #insertions + #deletions + #mismatches

Edit Distance in 2D Grid Representation

- Edit Graph

BAYLOR

Solving by Exhaustive Search or Greedy Algorithm

- Exhaustive Search Algorithm
- Greedy Algorithm

BAYLOR

Solving by Dynamic Programming

- Recursive Formula
- Dynamic Programming Algorithm

Overview

- Manhattan Tourist Problem
- Longest Common Subsequence Problem
- Edit Distance
- *Global Sequence Alignment*
- Local Sequence Alignment
- Alignment with Gap Penalty
- Multiple Sequence Alignment

from LCS and Edit Distance to Sequence Alignment

- LCS problem
  - Allows only insertions and deletions - no substitutions
  - Scores 1 for a match and 0 for an insertion or deletion
- Edit Distance problem
  - Allows insertions, deletions, and substitutions
  - Score 1 for an insertion or deletion or substitution and 0 for a match
- Sequence Alignment Problem
  - Allows gaps (insertions and deletions) and mismatches (substitutions)
  - Uses any scoring schemes

Formulation of Global Alignment Problem

- 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

Basic Scoring Scheme

- Simplest Scoring Scheme
  - Match premium: +α
  - Mismatch penalty: -μ
  - Insertion and deletion (gap) penalty: -σ

Score = α #matches – μ #mismatches – σ (#insertions + #deletions)

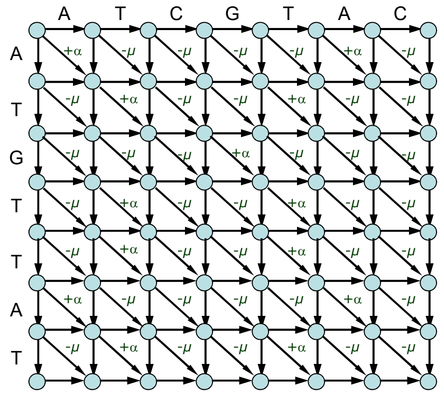
- Recursive Formula

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

BAYLOR

Example of Sequence Alignment

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



All horizontal and vertical edges: -σ

BAYLOR

Solving by Dynamic Programming

- Algorithm

```
GLOBALALIGNMENT($x, y$)
    $S_{0,0} \leftarrow 0$
    $for\ \ i \leftarrow 1\ \ to\ \ m$
        $S_{i,0} \leftarrow S_{i-1,0} - \sigma$
    $for\ \ j \leftarrow 1\ \ to\ \ n$
        $S_{0,j} \leftarrow S_{0,j-1} - \sigma$
    $for\ \ i \leftarrow 1\ \ to\ \ m$
        $for\ \ j \leftarrow 1\ \ to\ \ n$
            $if\ \ x_i = y_j$
                $S_{i,j} \leftarrow \max(S_{i-1,j} - \sigma, S_{i,j-1} - \sigma, S_{i-1,j-1} + \alpha)$
            $else$
                $S_{i,j} \leftarrow \max(S_{i-1,j} - \sigma, S_{i,j-1} - \sigma, S_{i-1,j-1} - \mu)$
    $return\ \ S_{m,n}$
```

BAYLOR

Advanced Scoring Schemes

- Percent Identity
  - Percentage of identical matches
- Percent Similarity
  - Percentage of nucleotide pairs with the same type
  - Percentage of similar amino acid pairs in biochemical structure
- Advanced Scoring Schemes
  - Varying scores in similarity
  - Varying, strong penalties for mismatches
  - Relative likelihood of evolutionary relationship

    → Probability of mutations
  - Define scoring matrix for DNA or protein sequences

BAYLOR

Scoring Matrices (1)

- Scoring Matrix δ
  - Also called substitution matrix
  - 4 × 4 array representation for DNA sequences
    or (4+1) × (4+1) array
  - 20 × 20 array representation for protein sequences
    or (20+1) × (20+1) array
  - Entry of δ (*i,j*) has the score between *i* and *j*,
    i.e., the rate at which *i* is substituted with *j* over time

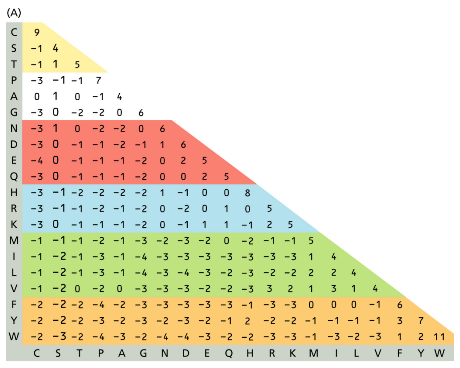
- Recursive Formula

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

BAYLOR

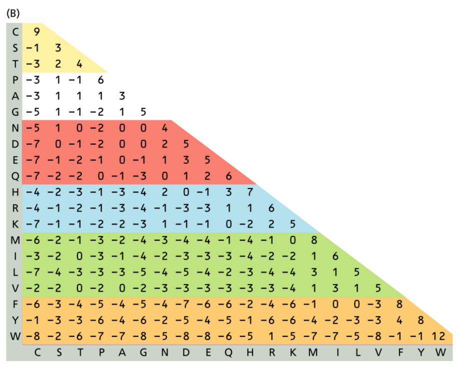
Scoring Matrices (2)

- Scoring Matrix Examples
  - BLOSUM-62
  - PAM120



(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



(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

BAYLOR

Overview

- Manhattan Tourist Problem
- Longest Common Subsequence Problem
- Edit Distance
- Global Sequence Alignment
- *Local Sequence Alignment*
- Alignment with Gap Penalty
- Multiple Sequence Alignment

BAYLOR

Local Sequence Alignment (1)

- Example
  - x = "TCAGTGTCGAAGTTA"
  - y = "TAGGCTAGCAGTGTG"
- Global 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
```

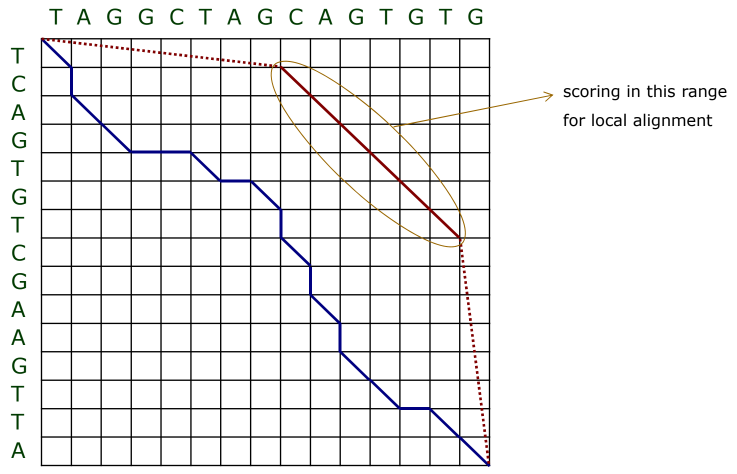
- Local Alignment

```
              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
```

BAYLOR

Local Sequence Alignment (2)

- Example - Continued


T A G G C T A G C A G T G T G
T C A G T G T C G A A G T T A
scoring in this range for local alignment

Global vs. Local Alignment

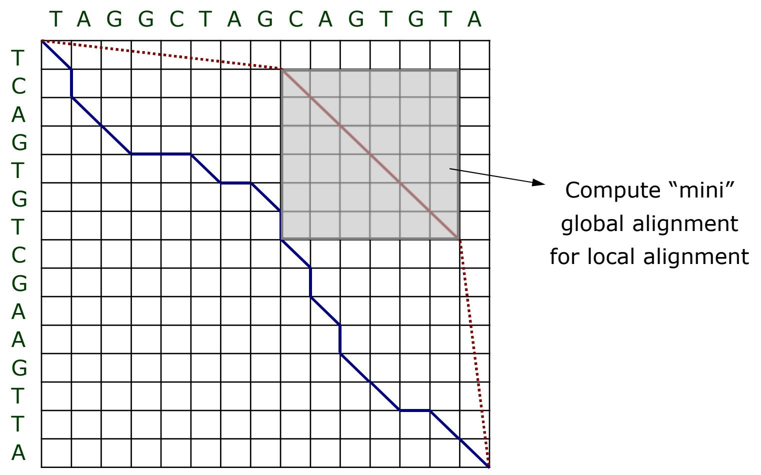
- Global Alignment Problem
  - Finds the path having the largest weight between vertices (0,0) and (*m*,*n*) in the edit graph
- Local Alignment Problem
  - Finds the path having the largest weight between two arbitrary vertices, (*i*,*j*) and (*i′*, *j′*), in the edit graph
- Score Comparison
  - The score of local alignment must be greater than (or equal to) the score of global alignment

Formulation of Local Alignment Problem

- 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 Alignment

- Strategy


T A G G C T A G C A G T G T A
T C A G T G T C G A A G T T A
Compute "mini" global alignment for local alignment

Solving by Exhaustive Search (1)

- 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 :

BAYLOR

Solving by Exhaustive Search (2)

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

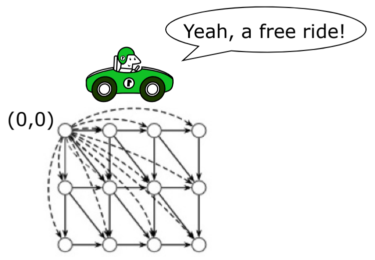
- Runtime
  - Suppose two sequences have the same length *n*
  - Global alignment :
  - Total runtime :

- Solution
  - Free ride !

BAYLOR

Solving by Dynamic Programming

- Free Ride
  - Assigns 0 weights from (0,0) to any other nodes (*i,j*)



- Recursive Formula

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

- Runtime ?

BAYLOR

Overview

- Manhattan Tourist Problem
- Longest Common Subsequence Problem
- Edit Distance
- Global Sequence Alignment
- Local Sequence Alignment
- *Alignment with Gap Penalty*
- Multiple Sequence Alignment

BAYLOR

Scoring Insertions/Deletions

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

    $\rightarrow$ too severe penalty for a series of 100 consecutive insertions/deletions
- Example
  - x="ATAGC", y="ATATTGC"

    single event

    ```
    ATA__GC
    ATATTGC
    ```
  - x="ATAGGC", y="ATGTGC"

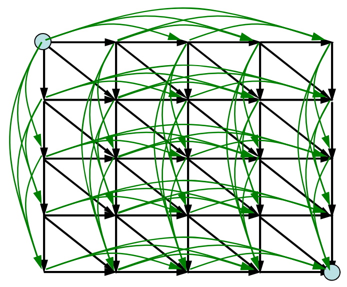
    ```
    ATAG_GC
    AT_GTGC
    ```

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 : $-(\rho + \sigma x)$
  - -ρ : gap opening penalty / -σ : 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 ?

BAYLOR

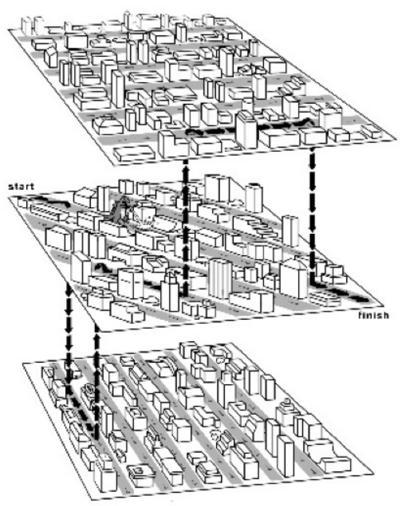
Improved Solution for Constant/Affine Gap Penalty

- 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 *x*
  - Lower layer for vertical edges
    - Creates/extends gaps in a sequence *y*

  - Gap opening penalty ($-\rho$) for jumping from middle layer to upper/lower layer
  - Gap extension penalty ($-\sigma$) for extending on upper/lower layer

BAYLOR

Example of 3-Layer Grid



upper layer (gaps in *x*)

main layer (matches/mismatches)

lower layer (gaps in *y*)

BAYLOR

Solving by Dynamic Programming

- Recursive Formula
  - Dynamic programming with recursion

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

$$S^{lower}_{i,j} = \max \begin{cases} S^{lower}_{i-1,j} - \sigma \\ S_{i-1,j} - (\rho + \sigma) \end{cases}$$ → continuing gap in *y*

→ starting gap in *y*

$$S^{upper}_{i,j} = \max \begin{cases} S^{upper}_{i,j-1} - \sigma \\ S_{i,j-1} - (\rho + \sigma) \end{cases}$$ → continuing gap in *x*

→ starting gap in *x*

  - Runtime ?

BAYLOR

Overview

- Manhattan Tourist Problem
- Longest Common Subsequence Problem
- Edit Distance
- Global Sequence Alignment
- Local Sequence Alignment
- Alignment with Gap Penalty
- *Multiple Sequence Alignment*

BAYLOR

Pairwise Alignment vs. Multiple Alignment

- Pairwise Alignment
  - Alignment of two sequences
  - Sometimes two sequences are functionally similar or have common ancestor although they have weak sequence similarity
- Multiple Alignment
  - Alignment of more than two sequences
  - Finds invisible similarity in pairwise alignment

BAYLOR

Alignment of 3 Sequences

- Alignment of 2 Sequences
  - Described in a 2-row representation
  - Best alignment is found in a 2-D matrix by dynamic programming
- Alignment of 3 Sequences
  - Described in a 3-row representation
  - x="ATGTG", y="ACGTA", z="ATCTG"

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

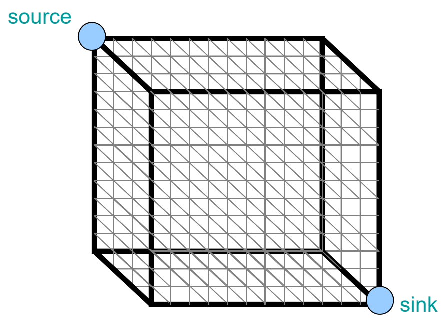
  - Best alignment is found in a 3-D matrix by dynamic programming

BAYLOR

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	–



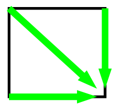
  - Path in 3-D grid :

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

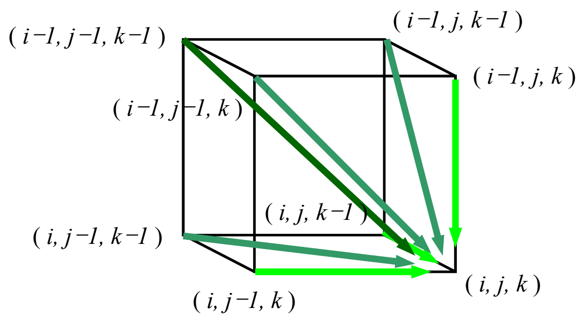
BAYLOR

3-D Grid Unit

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



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



Solving by Dynamic Programming

- Recursive 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 ?

Heuristics of Multiple Alignment

- Intuition
  - Implementing pairwise alignment (2-D alignment) many 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

Pairwise Alignment to Multiple Alignment

- Pairwise Alignments → Multiple Alignment

```
x: ACGCGG-C
y: ACGC-GAG
```

```
x: AC-GCGG-C
z: GCCGC-GAG
```

```
y: AC-GCGAG
z: GCCGCGAG
```

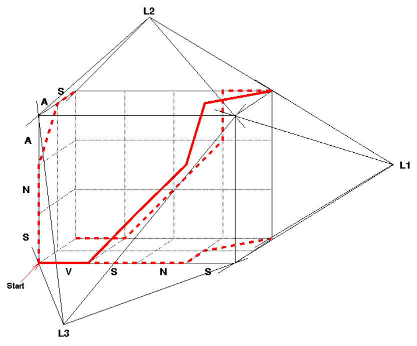
```
x: AC-GCGG-C
y: AC-GC-GAG
z: GCCGC-GAG
```

  - Can we construct a multiple alignment that induces pairwise alignments ?

BAYLOR

Multiple Alignment Projection

- Projection



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

BAYLOR

Greedy Approach (1)

- Process
  - (1) Implement pairwise alignment for all possible pairs of sequences
  - (2) Choose the most similar pair of sequences
  - (3) Merge them into a new sequence
  - (4) Choose the most similar sequence to the new sequence
  - (5) Repeat (3) and (4) until choosing all sequences

- Example
  - Step 1

```
s1: GATTCA          s2  GTCTGA      s1  GATTCA--
s2: GTCTGA          s4  GTCAGC      s4  G—T-CAGC
s3: GATATT    ⇒
s4: GTCAGC          s1  GAT-TCA     s2  G-TCTGA
                    s2  G-TCTGA     s3  GATAT-T

                    s1  GAT-TCA     s3  GAT-ATT
                    s3  GATAT-T     s4  G-TCAGC
```

Greedy Approach (2)

- Example - continued
  - Step 2

```
s2   GTCTGA   ⇒   s2,4  GTCt/aGa/c
s4   GTCAGC             ( called profile or consensus sequence )
```

  - Step 3

```
s1     GATTCA
s3     GATATT
s2,4   GTCt/aGa/c
```

- Features
  - *k*-way alignment (alignment of *k* sequences) → Runtime ?
  - Greedy algorithm → Not optimal multiple alignment

Scoring Schemes

- 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 alignment of sequences, a_i and a_j, imposed by a multiple alignment of k sequences
  - Denote the score of the pairwise alignment 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 (1)

- 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

Entropy-Based Scoring (2)

- Example

```
AAA
AAG
AAT
ATC
```

  - Frequency
    - 1st column: p(A) = 1, p(T) = p(G) = p(C) = 0
    - 2nd column: p(A) = 0.75, p(T) = 0.25, p(G) = p(C) = 0
    - 3rd column: p(A) = 0.25, p(T) = 0.25, p(C) = 0.25, p(G) = 0.25
  - Entropy

$$H\begin{pmatrix}A\\A\\A\\A\end{pmatrix}=0 \qquad H\begin{pmatrix}A\\A\\A\\T\end{pmatrix}=-\frac{3}{4}\log\frac{3}{4}-\frac{1}{4}\log\frac{1}{4}=0.244 \qquad H\begin{pmatrix}A\\G\\T\\C\end{pmatrix}=\left(-\frac{1}{4}\log\frac{1}{4}\right)\times 4=0.602$$

  - Entropy-based score in multiple alignment: 0 + 0.244 + 0.602

Questions?

- Lecture Slides are found on the Course Website,
  web.ecs.baylor.edu/faculty/cho/5330

BAYLOR