

EECS 4425:

Introductory Computational Bioinformatics

Fall 2018

Suprakash Datta

datta [at] cse.yorku.ca

Office: CSEB 3043

Phone: 416-736-2100 ext 77875

Course page: <http://www.cse.yorku.ca/course/4425>

Many of the slides are taken from www.bioalgorithms.info

Next: sequence alignment

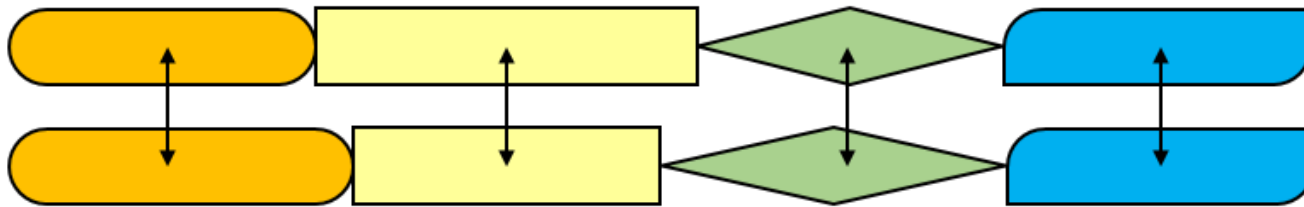
Why align?

Scarites	C	T	T	A	G	A	T	C	G	T	A	C	C	A	A	-	-	-	A	A	T	A	T	T	A	C
Carenum	C	T	T	A	G	A	T	C	G	T	A	C	C	A	C	A	-	T	A	C	-	T	T	T	A	C
Pasimachus	A	T	T	A	G	A	T	C	G	T	A	C	C	A	C	T	A	T	A	A	G	T	T	T	A	C
Pheropsophus	C	T	T	A	G	A	T	C	G	T	T	C	C	A	C	-	-	-	A	C	A	T	A	T	A	C
Brachinus armiger	A	T	T	A	G	A	T	C	G	T	A	C	C	A	C	-	-	-	A	T	A	T	A	T	T	C
Brachinus hirsutus	A	T	T	A	G	A	T	C	G	T	A	C	C	A	C	-	-	-	A	T	A	T	A	T	A	C
Aptinus	C	T	T	A	G	A	T	C	G	T	A	C	C	A	C	-	-	-	A	C	A	A	T	T	A	C
Pseudomorpha	C	T	T	A	G	A	T	C	G	T	A	C	C	-	-	-	-	-	A	C	A	A	A	T	A	C

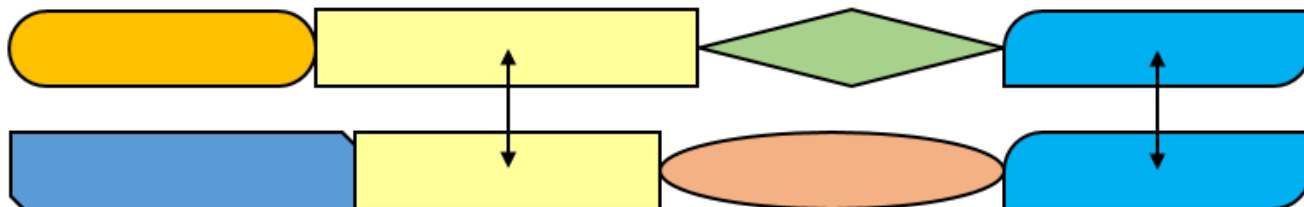
Picture from <http://www.sequence-alignment.com/>

Local vs Global Alignment

Picture from Wikipedia (By Yz cs5160 - Own work, CC BY-SA 4.0,
<https://commons.wikimedia.org/w/index.php?curid=54415549>)



Global Alignment



Local Alignment

Dynamic programming (DP)

- Typically used for optimization problems
- Often results in efficient algorithms
- Not applicable to all problems

Caveats:

- Need not yield poly-time algorithms
- No unique formulations for most problems
- May not rule out greedy algorithms

Example

- Counting the number of shortest paths in a grid
- Counting the number of shortest paths in a grid with blocked intersections
- Finding paths in a weighted grid
- Sequence alignment

Setting up DP in practice

- The optimal solution should be computable as a (recursive) function of the solution to sub-problems
- Solve sub-problems systematically and store solutions (to avoid duplication of work).

Number of paths in a grid

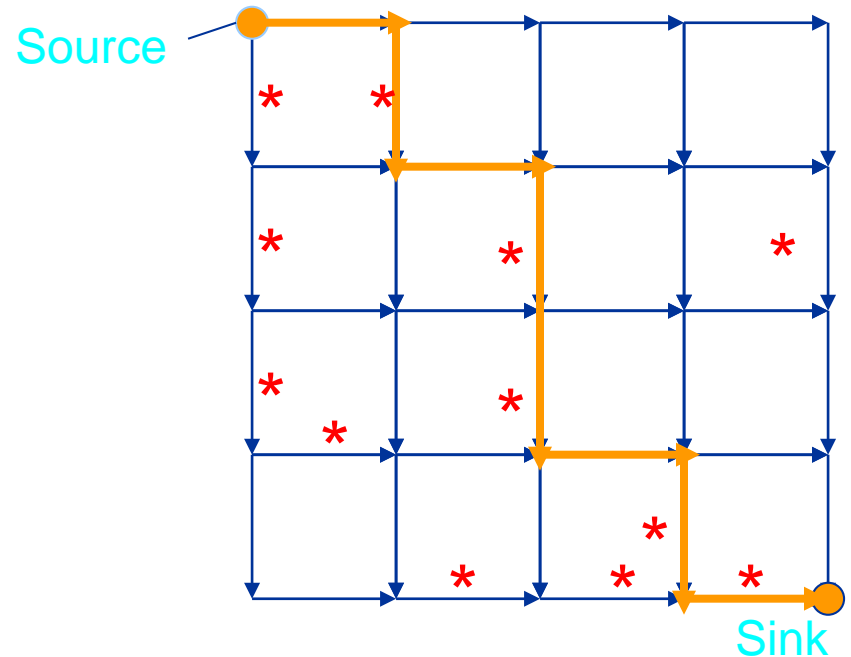
- Combinatorial approach
- DP approach: how can we decompose the problem into sub-problems ?

Number of paths in a grid with blocked intersections

- Combinatorial approach?
- DP approach: how can we decompose the problem into sub-problems ?

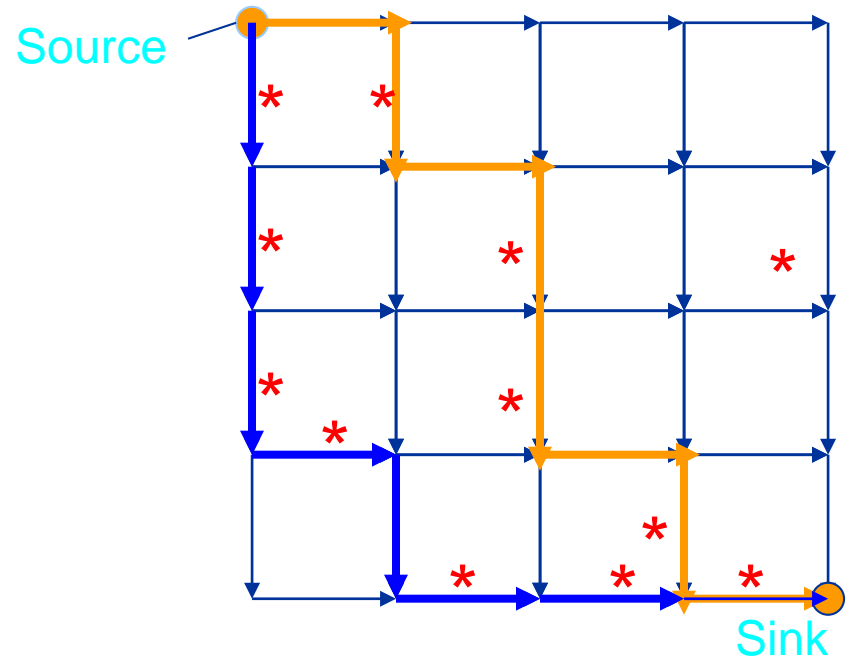
Manhattan Tourist Problem (MTP)

Imagine seeking a path (from source to sink) to travel (only eastward and southward) with the most number of attractions (*) in the Manhattan grid



Manhattan Tourist Problem (MTP)

Imagine seeking a path (from source to sink) to travel (only eastward and southward) with the most number of attractions (*) in the Manhattan grid



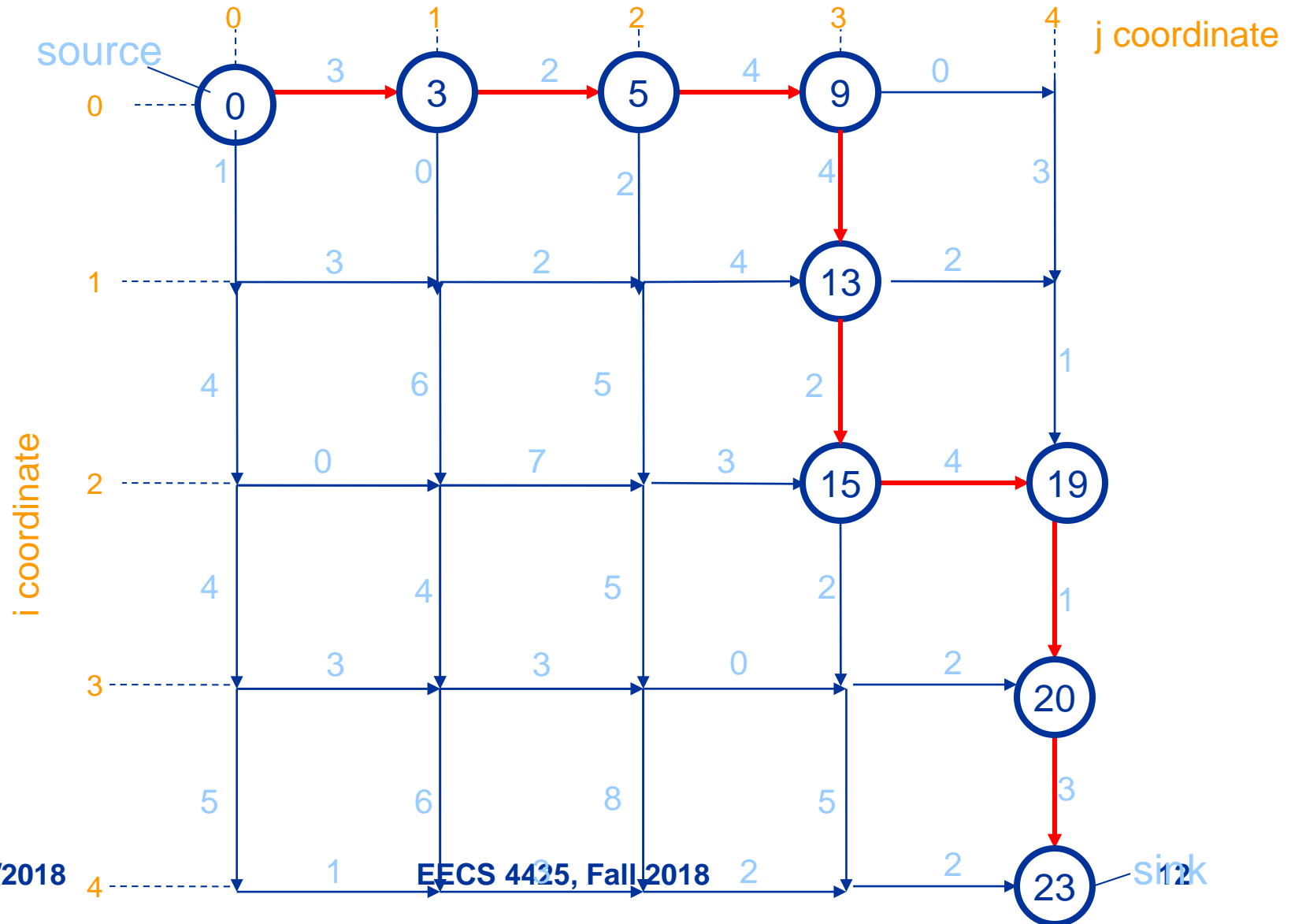
Manhattan Tourist Problem: Formulation

Goal: Find the best path in a weighted grid.

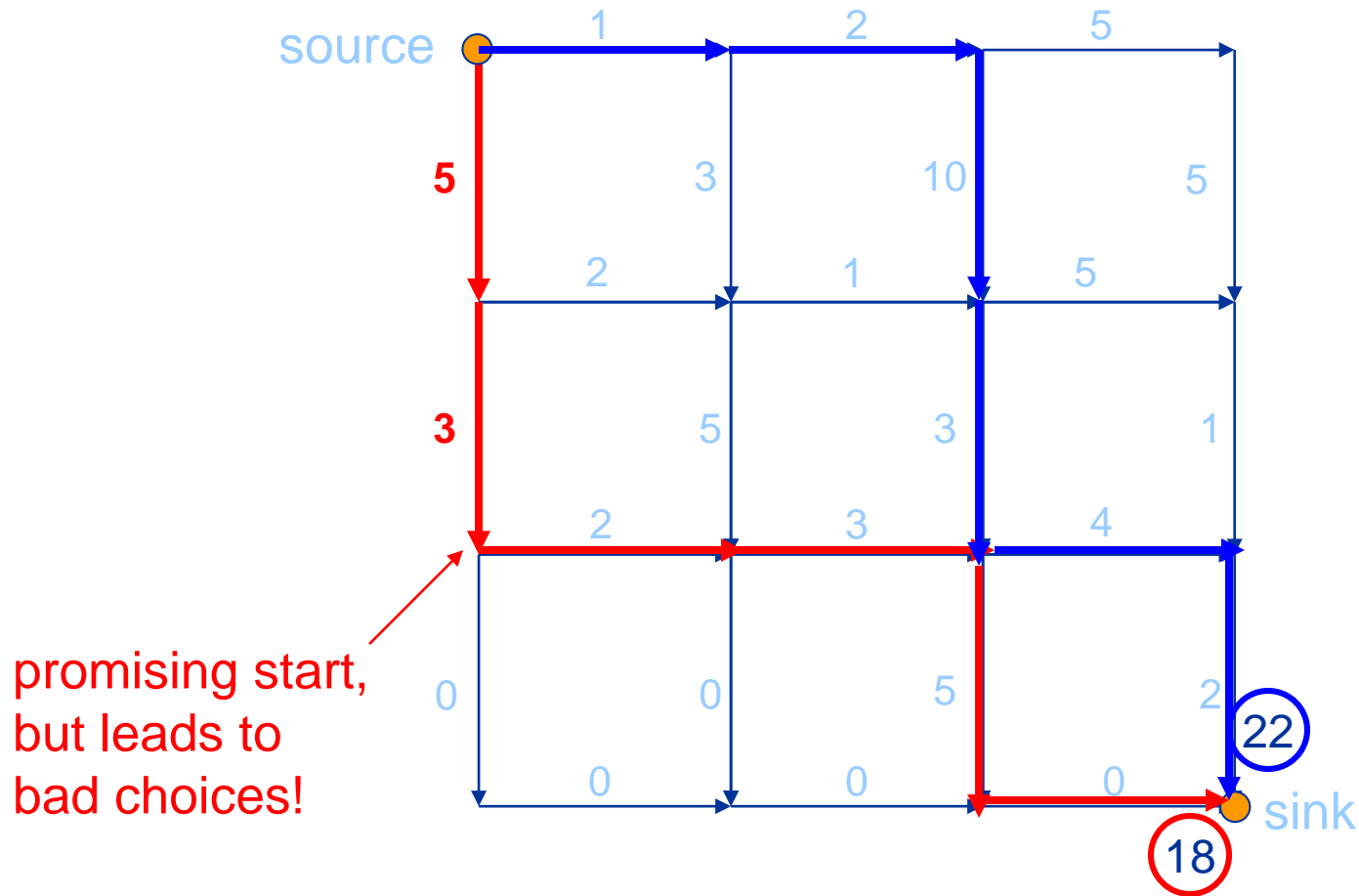
Input: A weighted grid \mathbf{G} with two distinct vertices, one labeled “*source*” and the other labeled “*sink*”

Output: A best path in \mathbf{G} from “*source*” to “*sink*”

MTP: An Example



MTP: Greedy Algorithm Is Not Optimal



MTP: Recurrence

Computing the score for a point (i,j) by the recurrence relation:

$$s_{i,j} = \max \left\{ \begin{array}{l} s_{i-1,j} + \text{weight of the edge between } (i-1, j) \text{ and } (i, j) \\ s_{i,j-1} + \text{weight of the edge between } (i, j-1) \text{ and } (i, j) \end{array} \right.$$

The running time is **$n \times m$** for a **n** by **m** grid
(**n** = # of rows, **m** = # of columns)

MTP: Simple Recursive Program

MT(n, m)

if $n=0$ or $m=0$

return $Line(n, m)$

$x \leftarrow MT(n-1, m) +$

length of the edge from $(n-1, m)$ to (n, m)

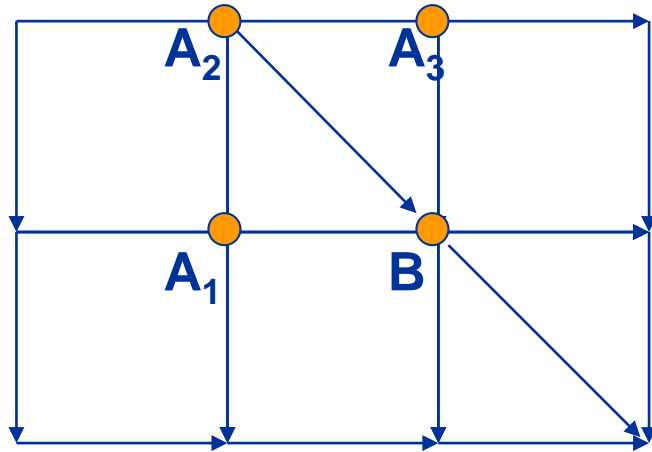
$y \leftarrow MT(n, m-1) +$

length of the edge from $(n, m-1)$ to (n, m)

return $\max\{x, y\}$

What's wrong with this approach?

Manhattan Is Not A Perfect Grid



What about diagonals?

- The score at point B is given by:

$$s_B = \max \left\{ \begin{array}{l} s_{A_1} + \text{weight of the edge } (A_1, B) \\ s_{A_2} + \text{weight of the edge } (A_2, B) \\ s_{A_3} + \text{weight of the edge } (A_3, B) \end{array} \right.$$

Manhattan Is Not A Perfect Grid (cont'd)

Computing the score for point x is given by the recurrence relation:

$$s_x = \max_{\text{of}} \left\{ s_y + \text{weight of vertex } (y, x) \text{ where } y \in \text{Predecessors}(x) \right.$$

- Predecessors (x) – set of vertices that have edges leading to x
- The running time for a graph $G(V, E)$ (V is the set of all vertices and E is the set of all edges) is $O(E)$ since each edge is evaluated once

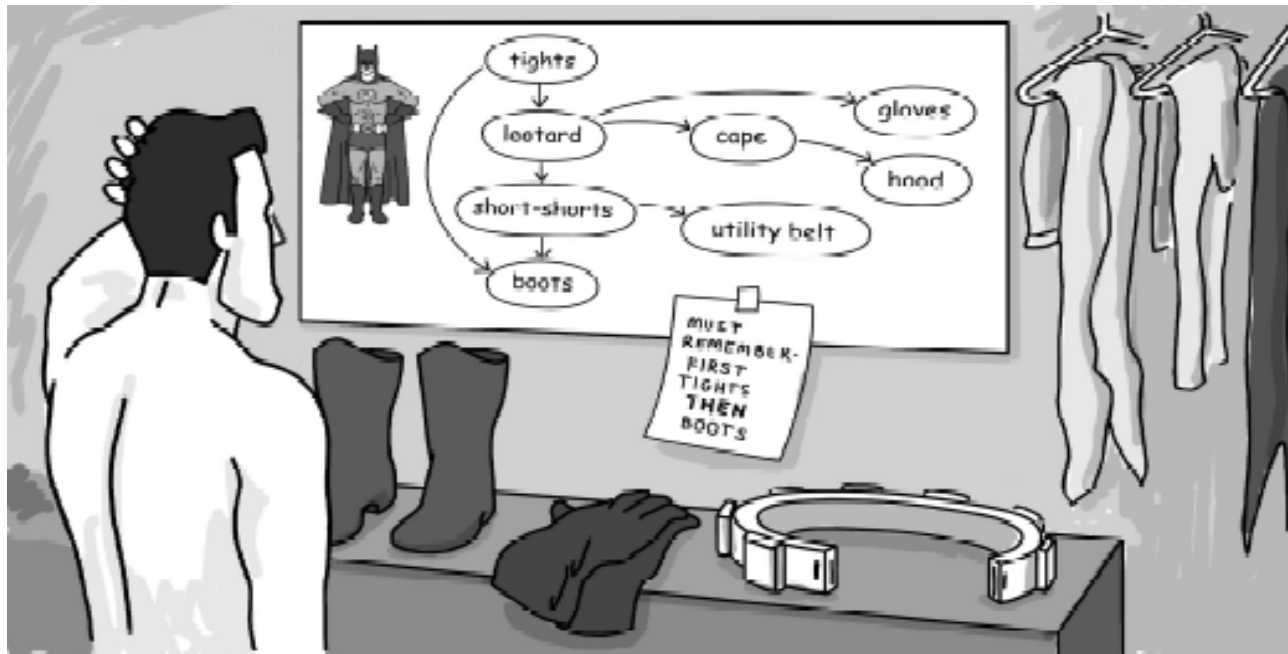
Traveling on the Grid

- The only hitch is that one must decide on the order in which visit the vertices
- By the time the vertex x is analyzed, the values s_y for all its predecessors y should be computed – otherwise we are in trouble.
- We need to traverse the vertices in some order
- Try to find such order for a directed cycle

???

DAG: Directed Acyclic Graph

- Since Manhattan is not a perfect regular grid, we represent it as a DAG
- DAG for *Dressing in the morning* problem

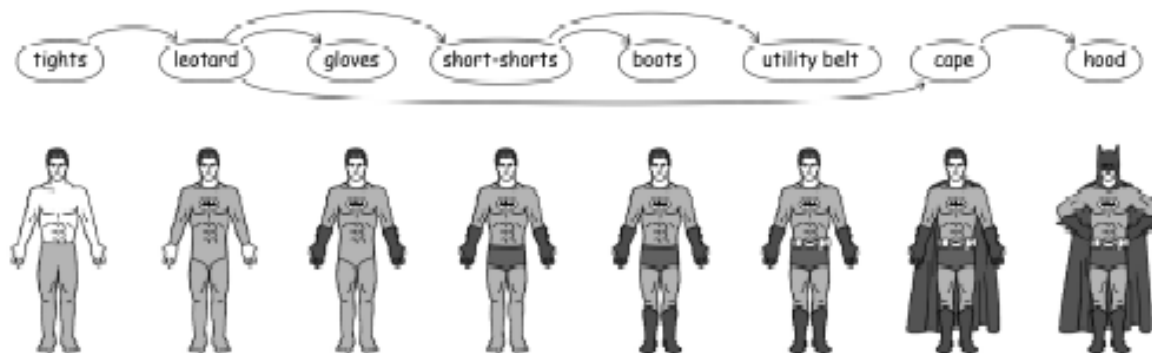
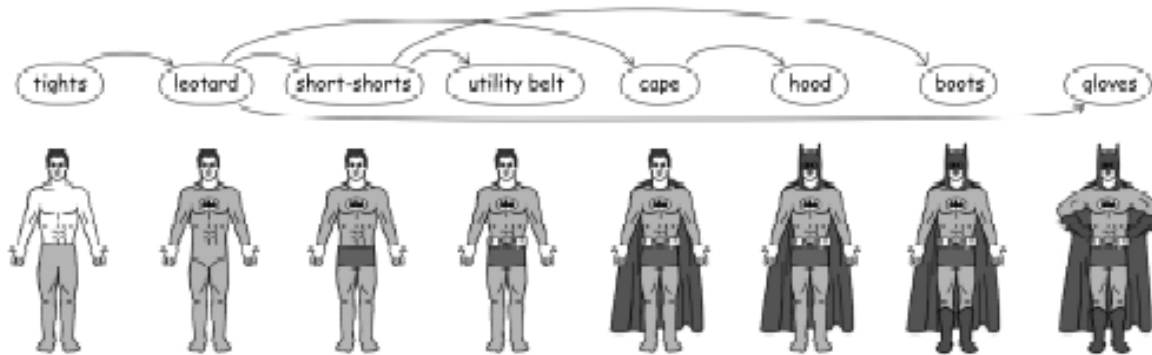


Topological Ordering

- A numbering of vertices of the graph is called ***topological ordering*** of the DAG if every edge of the DAG connects a vertex with a smaller label to a vertex with a larger label
- In other words, if vertices are positioned on a line in an increasing order of labels then all edges go from left to right.

Topological ordering

- 2 different topological orderings of the $DA \sim$



Longest Path in DAG Problem

- Goal: Find a longest path between two vertices in a weighted DAG
- Input: A weighted DAG G with source and sink vertices
- Output: A longest path in G from source to sink

Longest Path in DAG: Dynamic Programming

- Suppose vertex v has indegree 3 and predecessors $\{u_1, u_2, u_3\}$
- Longest path to v from source is:

$$s_v = \max_{\text{of}} \begin{cases} s_{u_1} + \text{weight of edge from } u_1 \text{ to } v \\ s_{u_2} + \text{weight of edge from } u_2 \text{ to } v \\ s_{u_3} + \text{weight of edge from } u_3 \text{ to } v \end{cases}$$

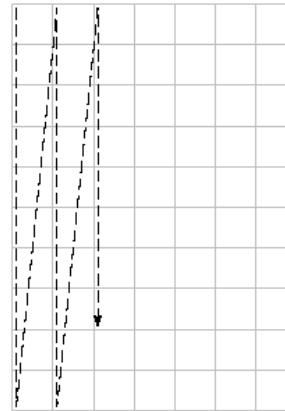
In General:

$$s_v = \max_u (s_u + \text{weight of edge from } \mathbf{u} \text{ to } \mathbf{v})$$

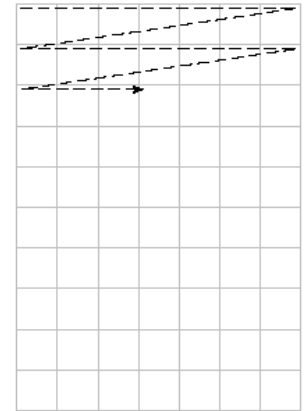
Traversing the Manhattan Grid

- 3 different strategies:
 - a) Column by column
 - b) Row by row
 - c) Along diagonals

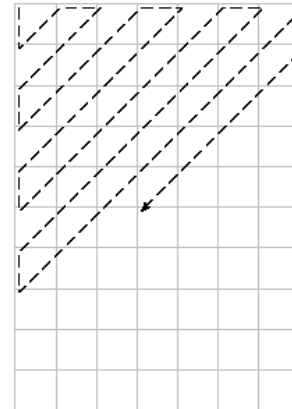
a)



b)



c)



Alignment: 2 row representation

Given 2 DNA sequences v and w :

v : **A**T**C**T**G**A**T** $m = 7$
 w : **T****G****C**A**T****A** $n = 6$

Alignment : $2 * k$ matrix ($k > m, n$)

letters of v

letters of w

A	T	--	G	T	T	A	T	--
A	T	C	G	T	--	A	--	C

4 matches

2 insertions

2 deletions

Aligning DNA Sequences

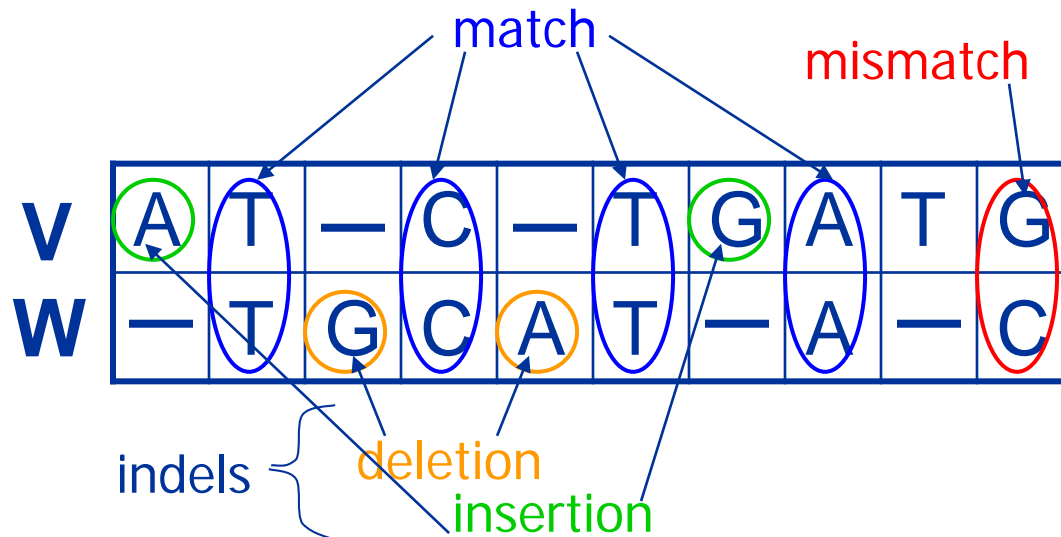
V = ATCTGATG

$n = 8$

W = TGCATAC

$m = 7$

4 matches
1 mismatches
2 insertions
2 deletions



Aligning DNA Sequences - 2

- Brute force is infeasible....
- Number of alignments of $X[1..n], Y[1..m]$,
 $n < m$ is $\binom{m+n}{n}$
- For $m=n$, this is about $2^{2n}/\pi n$

Longest Common Subsequence (LCS) – Alignment without Mismatches

- Given two sequences

$$\mathbf{v} = v_1 v_2 \dots v_m \text{ and } \mathbf{w} = w_1 w_2 \dots w_n$$

- The LCS of \mathbf{v} and \mathbf{w} is a sequence of positions in

$$\mathbf{v}: 1 \leq i_1 < i_2 < \dots < i_t \leq m$$

and a sequence of positions in

$$\mathbf{w}: 1 \leq j_1 < j_2 < \dots < j_t \leq n$$

such that i_t -th letter of \mathbf{v} equals to j_t -letter of \mathbf{w} and t is maximal

LCS: Example

<i>i</i> coords:	0	1	2	2	3	3	4	5	6	7	8
elements of <i>v</i>	A	T	--	C	--	T	G	A	T	C	
elements of <i>w</i>	--	T	G	C	A	T	--	A	--	C	
<i>j</i> coords:	0	0	1	2	3	4	5	5	6	6	7

$(0,0) \rightarrow (1,0) \rightarrow (2,1) \rightarrow (2,2) \rightarrow (3,3) \rightarrow (3,4) \rightarrow (4,5) \rightarrow (5,5) \rightarrow (6,6) \rightarrow (7,6) \rightarrow (8,7)$

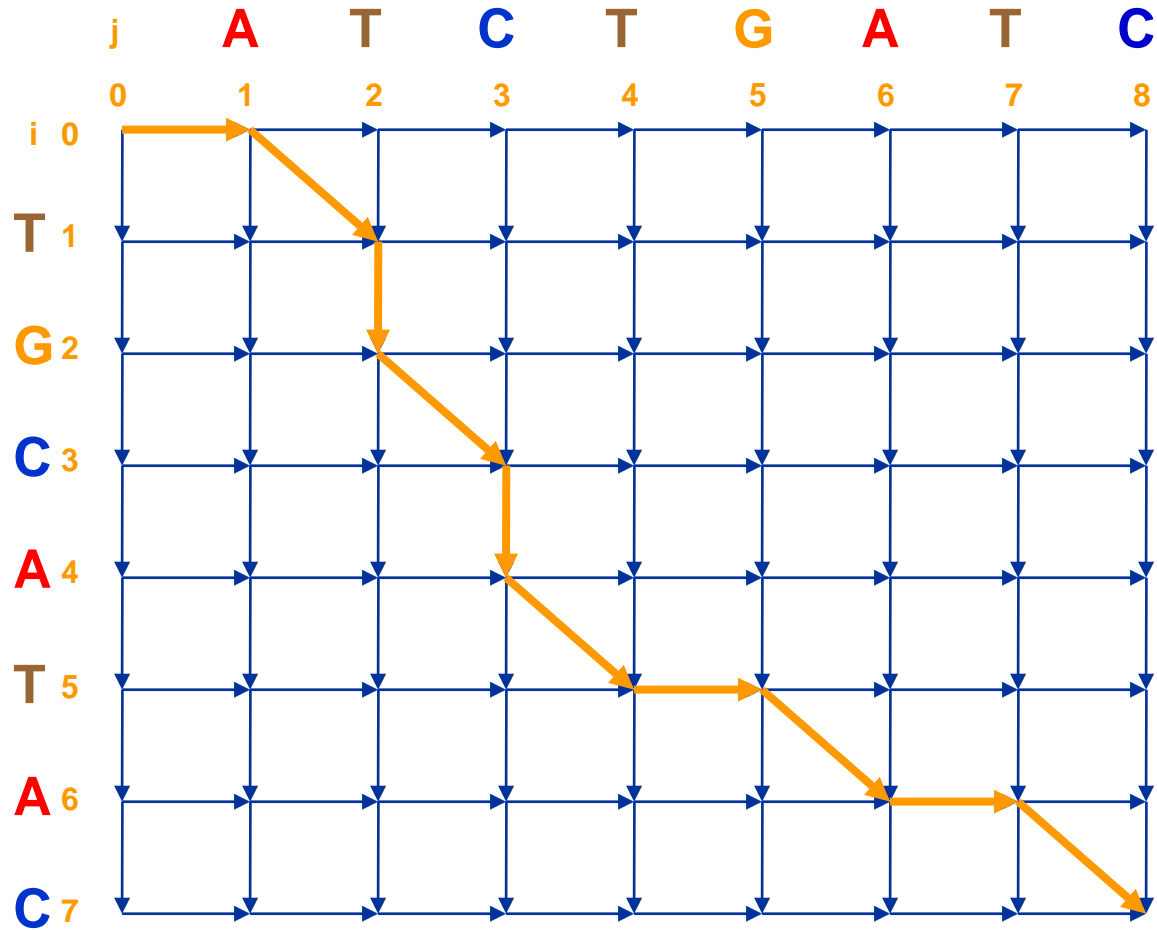
Matches shown in red

positions in *v*: 2 < 3 < 4 < 6 < 8

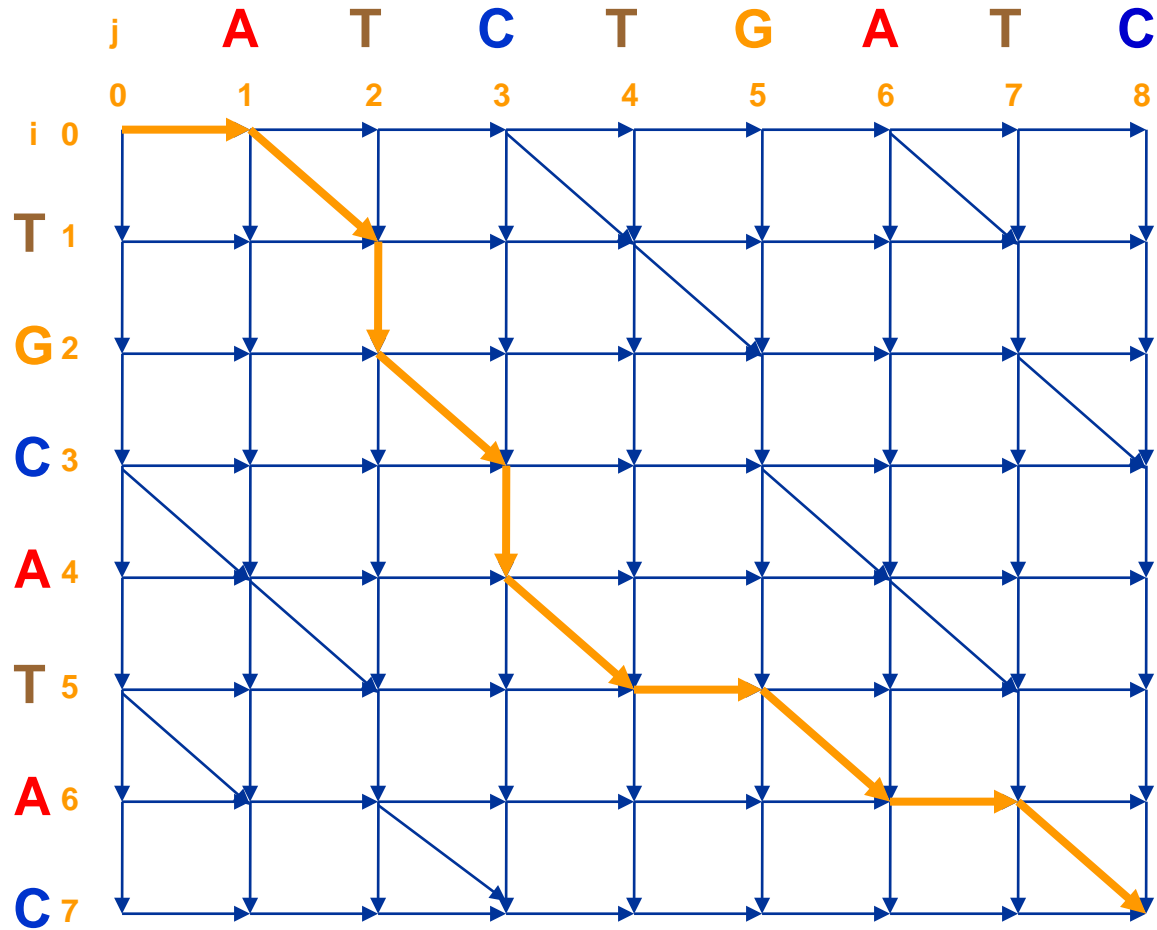
positions in *w*: 1 < 3 < 5 < 6 < 7

Every common subsequence is a path in 2-D grid

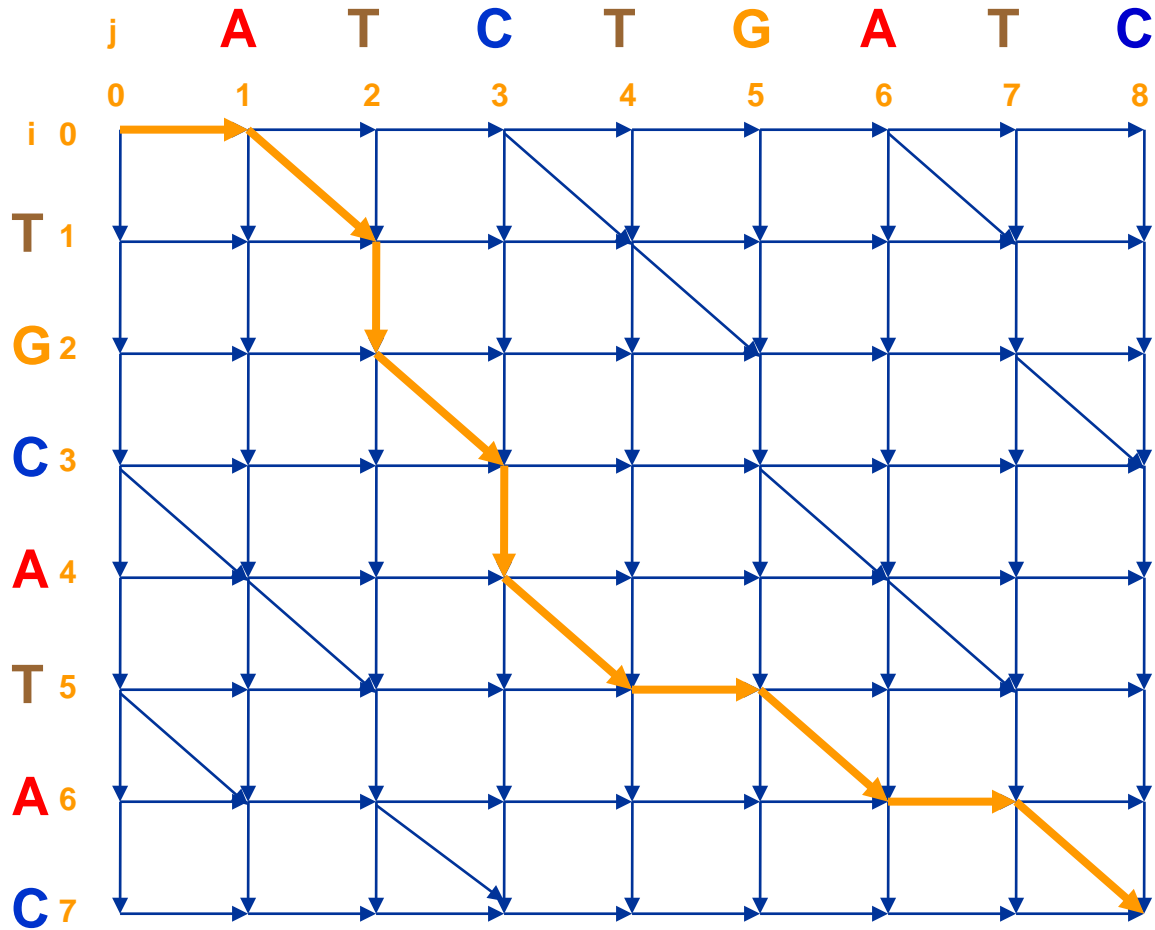
LCS Problem as Manhattan Tourist Problem



Edit Graph for LCS Problem



Edit Graph for LCS Problem



Every path is a common subsequence.

Every diagonal edge adds an extra element to common subsequence

LCS Problem:
Find a path with maximum number of diagonal edges

Computing LCS

Let \mathbf{v}_i = prefix of \mathbf{v} of length i : $v_1 \dots v_i$

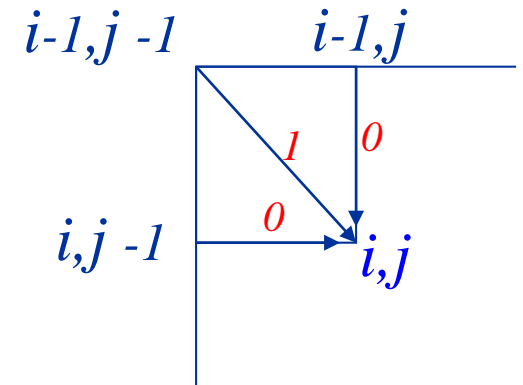
and \mathbf{w}_j = prefix of \mathbf{w} of length j : $w_1 \dots w_j$

The length of $\text{LCS}(\mathbf{v}_i, \mathbf{w}_j)$ is computed by:

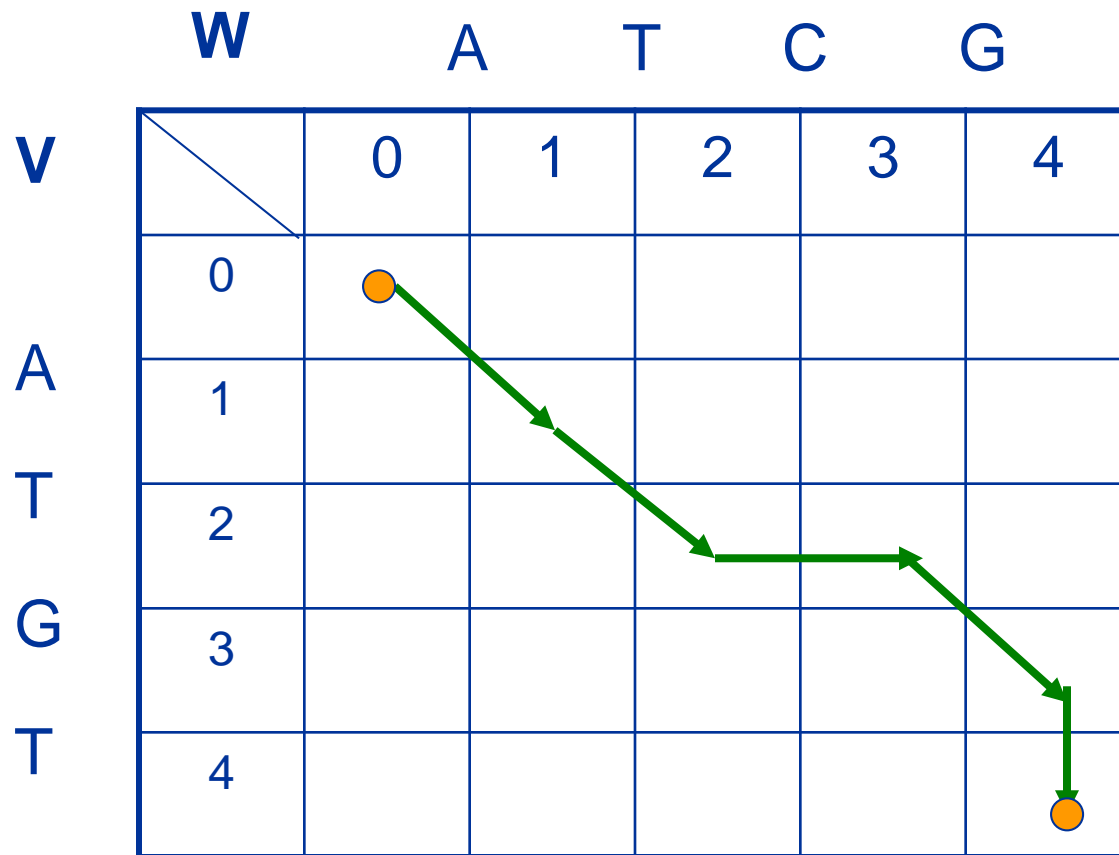
$$s_{i,j} = \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1 \text{ if } v_i = w_j \end{cases}$$

Computing LCS (cont'd)

$$s_{i,j} = \text{MAX} \begin{cases} s_{i-1,j} + 0 \\ s_{i,j-1} + 0 \\ s_{i-1,j-1} + 1, \end{cases} \quad \text{if } v_i = w_j$$



Every Path in the Grid Corresponds to an Alignment



$\swarrow \swarrow \rightarrow \swarrow \downarrow$
 0 1 2 2 3 4
 V = A T - G T
 | | |
 W = A T C G -
 0 1 2 3 4 4

Aligning Sequences without Insertions and Deletions: Hamming Distance

Given two DNA sequences \mathbf{v} and \mathbf{w} :

\mathbf{v} : A T A T A T A T
 \mathbf{w} : T A T A T A T A

- The Hamming distance: $d_H(\mathbf{v}, \mathbf{w}) = 8$ is large but the sequences are very similar

Aligning Sequences with Insertions and Deletions

By shifting one sequence over one position:

v : A T A T A T --
 w : -- T A T A T A

- The edit distance: $d_H(v, w) = 2$.
- Hamming distance neglects insertions and deletions in DNA

Edit Distance

Levenshtein (1966) introduced edit distance between two strings as the minimum number of elementary operations (insertions, deletions, and substitutions) to transform one string into the other

$d(\mathbf{v}, \mathbf{w})$ = MIN number of elementary operations
to transform $\mathbf{v} \rightarrow \mathbf{w}$

Edit Distance vs Hamming Distance

Hamming distance
always compares

i^{th} letter of \mathbf{v} with

i^{th} letter of \mathbf{w}

$\mathbf{v} = \text{ATATATAT}$
 | | | | | | | |
 $\mathbf{w} = \text{TATATATA}$

Hamming distance:

$$d(\mathbf{v}, \mathbf{w}) = 8$$

Computing Hamming distance
is a trivial task.

Edit Distance vs Hamming Distance

Hamming distance
always compares

i^{th} letter of \mathbf{v} with

i^{th} letter of \mathbf{w}

$\mathbf{v} = \text{ATATATAT}$
 | | | | | | | |
 $\mathbf{w} = \text{TATATATA}$

Just one shift
— — — — — →
Make it all line up

Hamming distance:

$$d(\mathbf{v}, \mathbf{w}) = 8$$

Computing Hamming distance
is a **trivial** task

Edit distance
may compare

i^{th} letter of \mathbf{v} with

j^{th} letter of \mathbf{w}

$\mathbf{v} = - \text{ATATATAT}$
 | | | | | | | |
 $\mathbf{w} = \text{TATATATA}$

Edit distance:

$$d(\mathbf{v}, \mathbf{w}) = 2$$

Computing edit distance
is a non-trivial task

Edit Distance vs Hamming Distance

Hamming distance
always compares

i^{th} letter of \mathbf{v} with
 i^{th} letter of \mathbf{w}

$\mathbf{v} = \text{ATATATAT}$
 | | | | | | | |
 $\mathbf{w} = \text{TATATATA}$

Hamming distance:

$$d(\mathbf{v}, \mathbf{w}) = 8$$

Edit distance
may compare

i^{th} letter of \mathbf{v} with
 j^{th} letter of \mathbf{w}

$\mathbf{v} = - \text{ATATATAT}$
 | | | | | | | |
 $\mathbf{w} = \text{TATATATA}$

Edit distance:

$$d(\mathbf{v}, \mathbf{w}) = 2$$

(one insertion and one deletion)

How to find what j goes with what i ???

Edit Distance: Example

TGCATAT → ATCCGAT in 5 steps

TGCATAT^T → (delete last ^T)

TGCAT^A → (delete last ^A)

TGCAT → (insert A at front)

AT^CCAT → (substitute ^C for 3rd ^G)

AT^CCAT → (insert G before last A)

ATCCGAT (Done)

Edit Distance: Example

TGCATAT → ATCCGAT in 5 steps

TGCATAT^T → (delete last ^T)

TGCAT^A → (delete last ^A)

TGCAT → (insert A at front)

AT^CCAT → (substitute ^C for 3rd ^G)

AT^CCAT → (insert G before last A)

ATCCGAT (Done)

What is the edit distance? 5?

Edit Distance: Example (cont'd)

TGCATAT → ATCCGAT in 4 steps

TGCATAT → (insert A at front)

ATGCATAT → (delete 6th T)

ATGCATA → (substitute G for 5th A)

ATGCGTA → (substitute C for 3rd G)

ATCCGAT (Done)

Edit Distance: Example (cont'd)

TGCATAT → ATCCGAT in 4 steps

TGCATAT → (insert A at front)

ATGCATAT → (delete 6th T)

ATGCATA → (substitute G for 5th A)

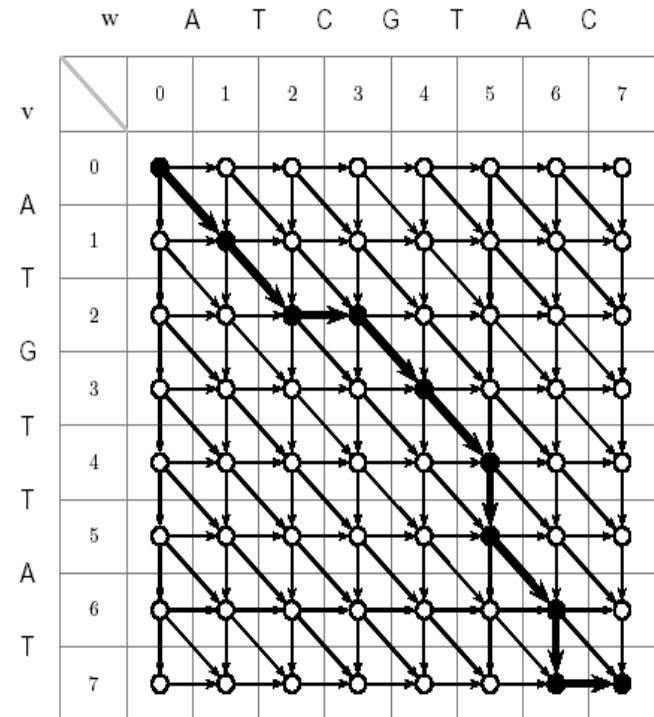
ATGCGTA → (substitute C for 3rd G)

ATCCGAT (Done)

Can it be done in 3 steps???

The Alignment Grid

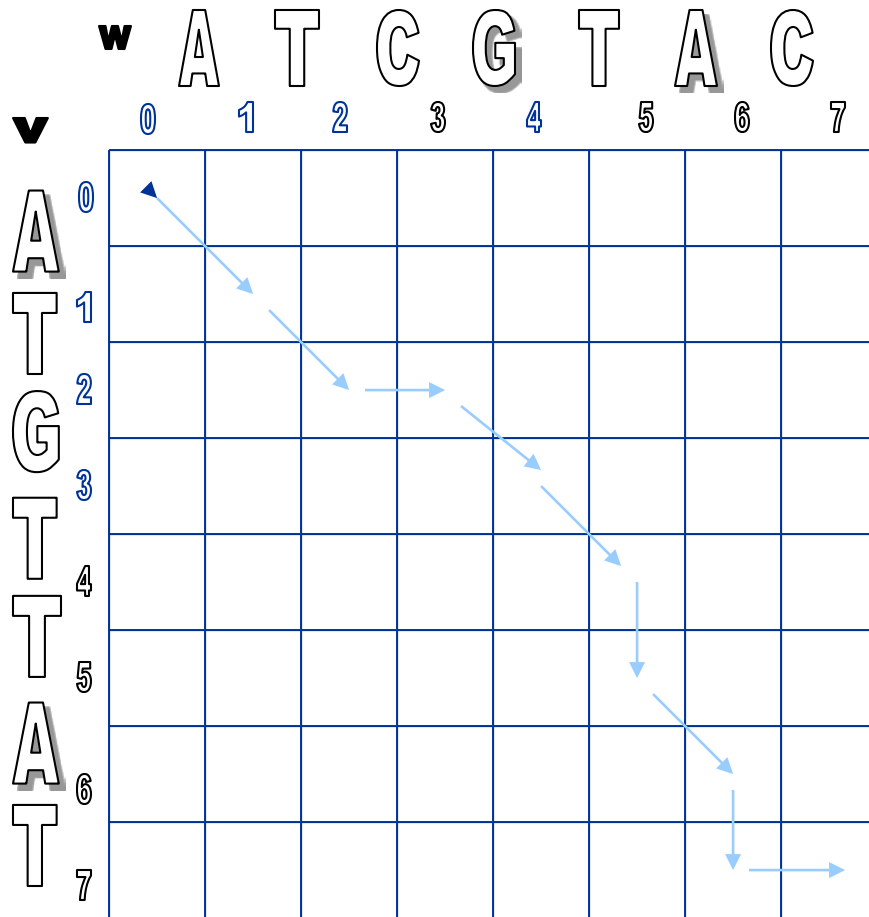
- Every alignment path is from source to sink

$$\begin{array}{rcccccccc}
 & 0 & 1 & 2 & 2 & 3 & 4 & 5 & 6 & 7 & 7 \\
 v & = & & A & T & - & G & T & T & A & T & - \\
 & & & | & | & & | & | & & | & & \\
 w & = & & A & T & C & G & T & - & A & - & C \\
 & & 0 & 1 & 2 & 3 & 4 & 5 & 5 & 6 & 6 & 7
 \end{array}$$


Legend for alignment paths:

\swarrow	\searrow	\rightarrow	\swarrow	\searrow	\downarrow	\swarrow	\downarrow	\rightarrow
A	T	-	G	T	T	A	T	-
A	T	C	G	T	-	A	-	C

Alignment as a Path in the Edit Graph

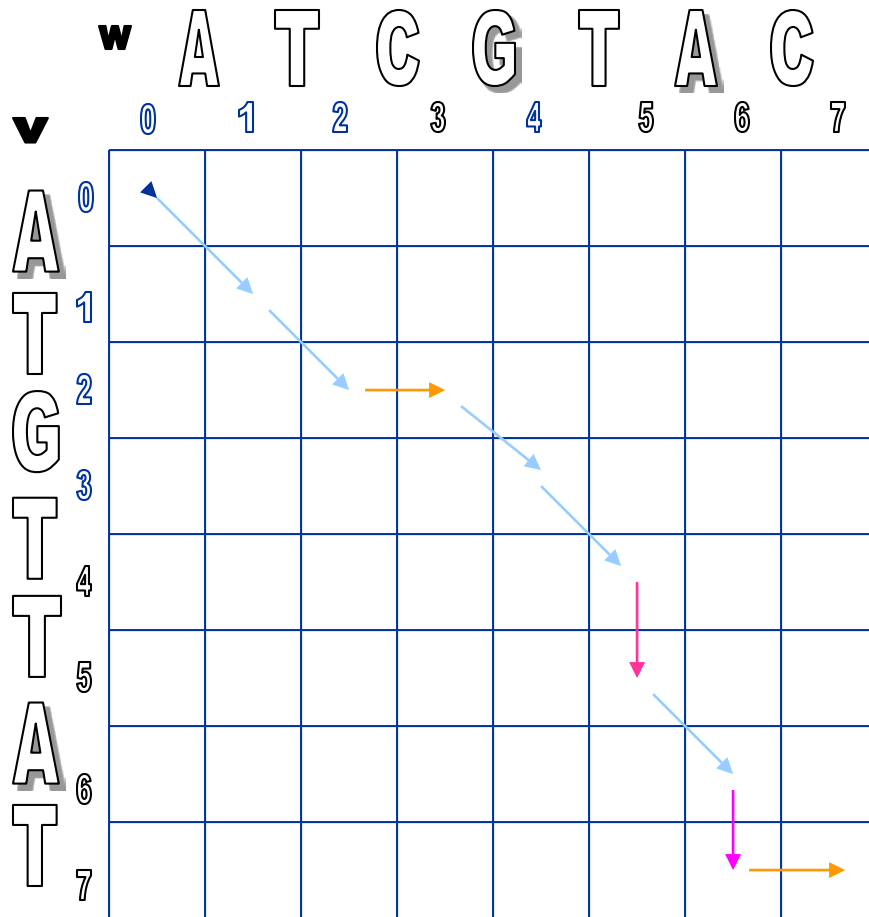


0	1	2	2	3	4	5	6	7	7
	A	T	_	G	T	T	A	T	_
	A	T	C	G	T	_	A	_	C
0	1	2	3	4	5	5	6	6	7

- Corresponding path -

(0, 0) , (1, 1) , (2, 2) , (2, 3) ,
 (3, 4) , (4, 5) , (5, 5) , (6, 6) ,
 (7, 6) , (7, 7)

Alignments in Edit Graph (cont'd)

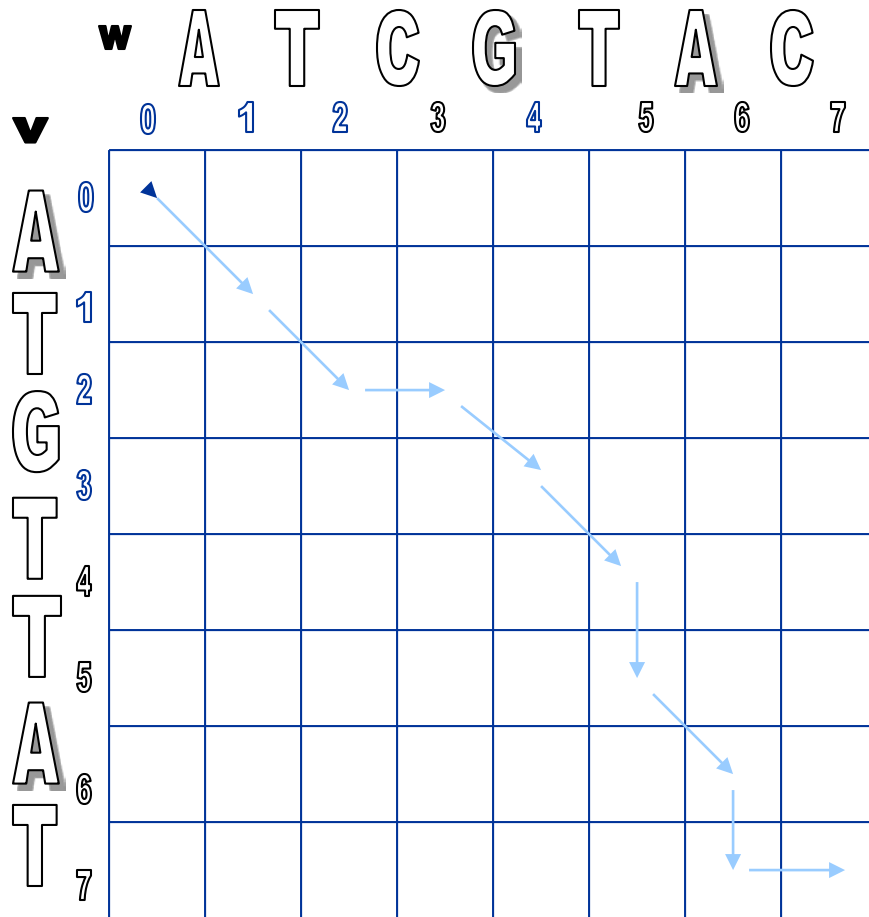


↓ and → represent indels in **v** and **w** with score 0.

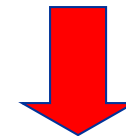
↘ represent matches with score 1.

- The score of the alignment path is 5.

Alignment as a Path in the Edit Graph

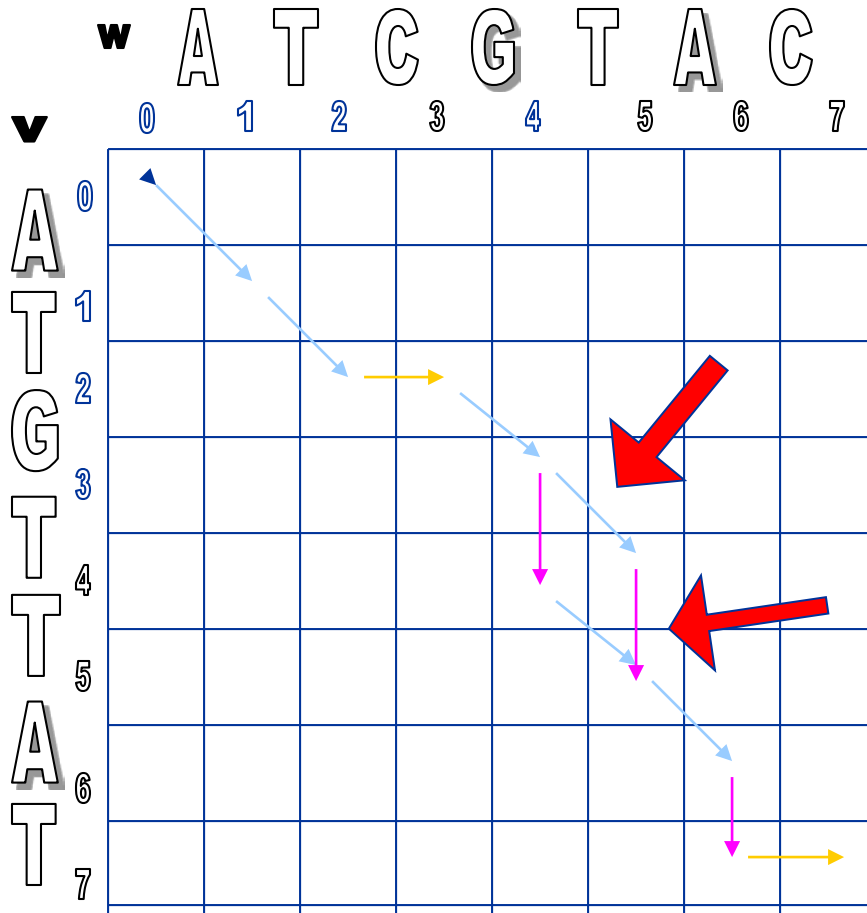


Every path in the edit graph corresponds to an alignment:



\	\	→	\	\	↓	\	↓	→
A	T	-	G	T	T	A	T	-
A	T	C	G	T	-	A	-	C

Alignment as a Path in the Edit Graph



Old Alignment

01223**4**5677

v= AT_G**T**TAT_

w= ATCG**T**_A_C

01234**5**5667

New Alignment

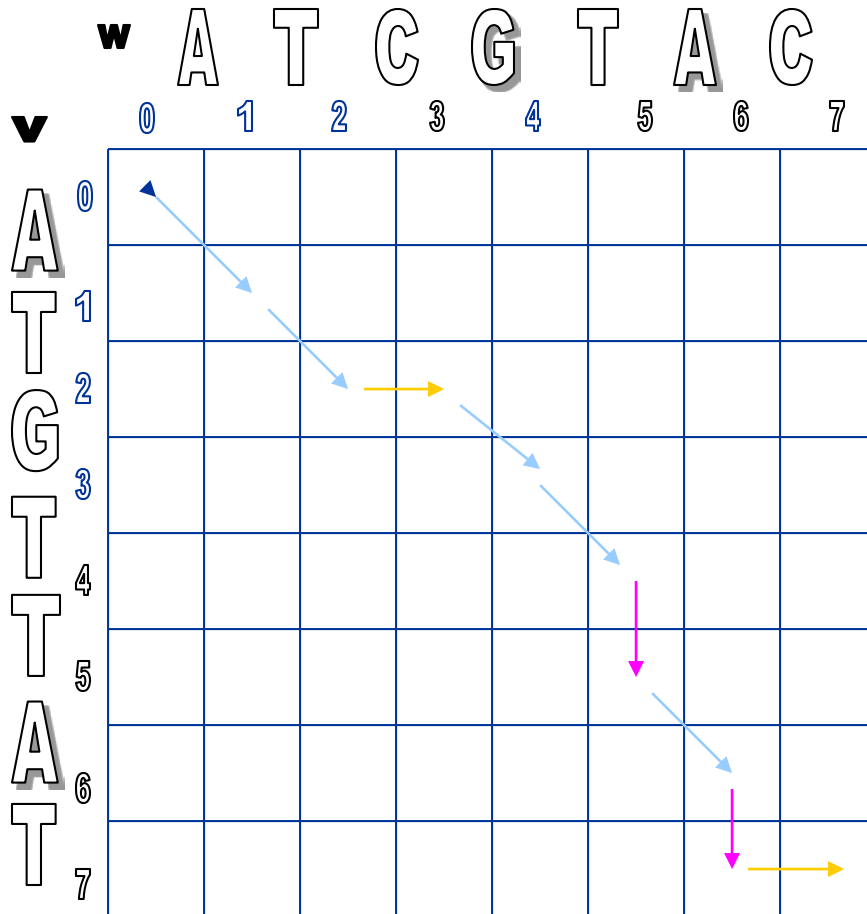
01223**4**5677

v= AT_G**T**TAT_

w= ATCG_**T**A_C

01234**4**5667

Alignment as a Path in the Edit Graph



012**2**345677

v= AT_GTTAT_

w= ATCGT_A_C

012**3**45**5**6**6**7

(0,0) , (1,1) , (2,2), (2,3),
 (3,4), (4,5), (5,5), (6,6),
 (7,6), (7,7)

Alignment: Dynamic Programming

$$S_{i,j} = \max \left\{ \begin{array}{l} S_{i-1,j-1} + 1 \text{ if } v_i = w_j \\ S_{i-1,j} \\ S_{i,j-1} \end{array} \right.$$

The diagram illustrates the recurrence relation for sequence alignment. The expression $S_{i,j} = \max \left\{ \begin{array}{l} S_{i-1,j-1} + 1 \text{ if } v_i = w_j \\ S_{i-1,j} \\ S_{i,j-1} \end{array} \right.$ is shown. A blue arrow points from the first term $S_{i-1,j-1} + 1$ to the right. A pink arrow points from the second term $S_{i-1,j}$ downwards. A yellow arrow points from the third term $S_{i,j-1}$ to the right.

Dynamic Programming Example

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

Initialize 1st row and
1st column to be all
zeroes.

Or, to be more
precise, initialize 0th
row and 0th column to
be all zeroes.

Dynamic Programming Example

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

$$S_{i,j} = \max \begin{cases} S_{i-1,j-1} & \leftarrow \text{value from NW} + 1, \text{ if } v_i = w_j \\ S_{i-1,j} & \leftarrow \text{value from North (top)} \\ S_{i,j-1} & \leftarrow \text{value from West (left)} \end{cases}$$

Alignment: tracing paths

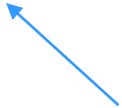
Arrows  show where the score originated from.



if from the top

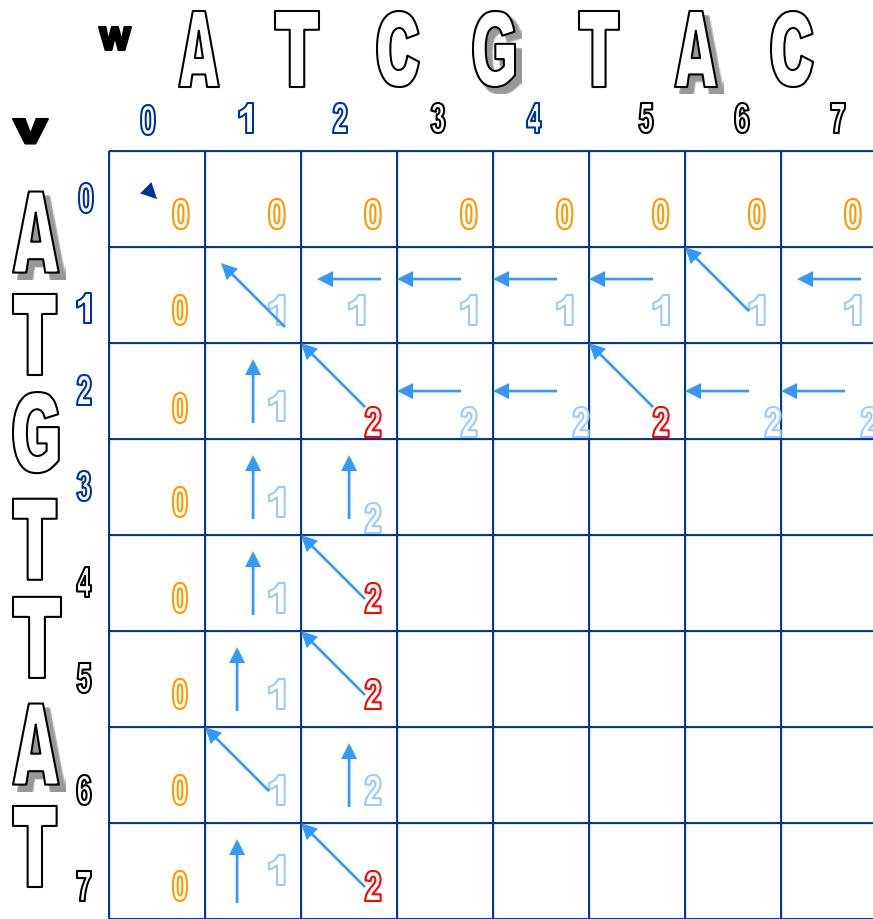


if from the left



if $v_i = w_j$

Path tracing example



Find a match in row and column 2.

$i=2, j=2,5$ is a match (T).

$j=2, i=4,5,7$ is a match (T).

Since $v_i = w_j$, $s_{i,j} = s_{i-1,j-1} + 1$

$$s_{2,2} = [s_{1,1} = 1] + 1$$

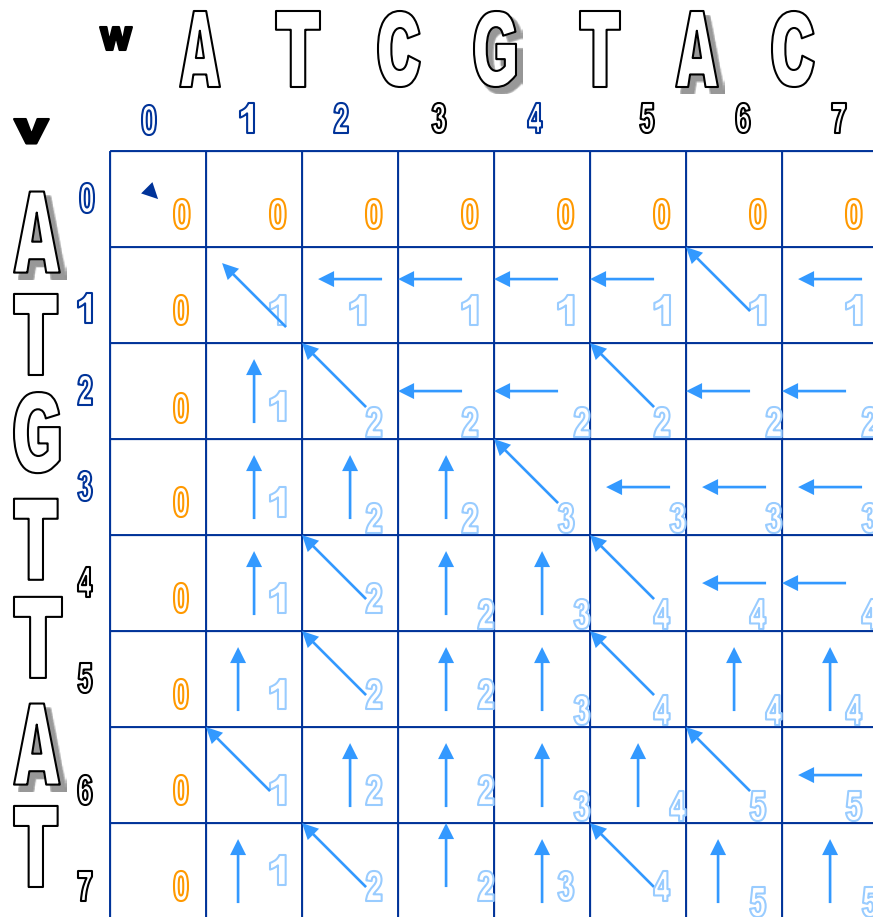
$$s_{2,5} = [s_{1,4} = 1] + 1$$

$$s_{4,2} = [s_{3,1} = 1] + 1$$

$$s_{5,2} = [s_{4,1} = 1] + 1$$

$$s_{7,2} = [s_{6,1} = 1] + 1$$

Path tracing example



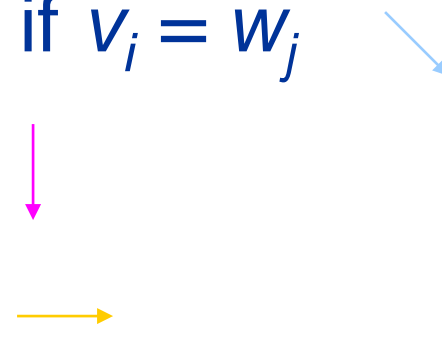
Continuing with the dynamic programming algorithm gives this result.

Alignment: Dynamic Programming

$$s_{i,j} = \max \left\{ \begin{array}{l} s_{i-1,j-1} + 1 \text{ if } v_i = w_j \\ s_{i-1,j} \\ s_{i,j-1} \end{array} \right.$$

The diagram illustrates the recurrence relation for sequence alignment. The expression $s_{i,j} = \max \left\{ \begin{array}{l} s_{i-1,j-1} + 1 \text{ if } v_i = w_j \\ s_{i-1,j} \\ s_{i,j-1} \end{array} \right.$ is shown. A blue arrow points from the first term $s_{i-1,j-1} + 1$ to the right. A pink arrow points from the second term $s_{i-1,j}$ downwards. A yellow arrow points from the third term $s_{i,j-1}$ to the right.

Alignment: Dynamic Programming

$$s_{i,j} = \max \left\{ \begin{array}{l} s_{i-1,j-1} + 1 \text{ if } v_i = w_j \\ s_{i-1,j} + 0 \\ s_{i,j-1} + 0 \end{array} \right.$$


This recurrence corresponds to the Manhattan Tourist problem (three incoming edges into a vertex) with all horizontal and vertical edges weighted by zero.

LCS Algorithm

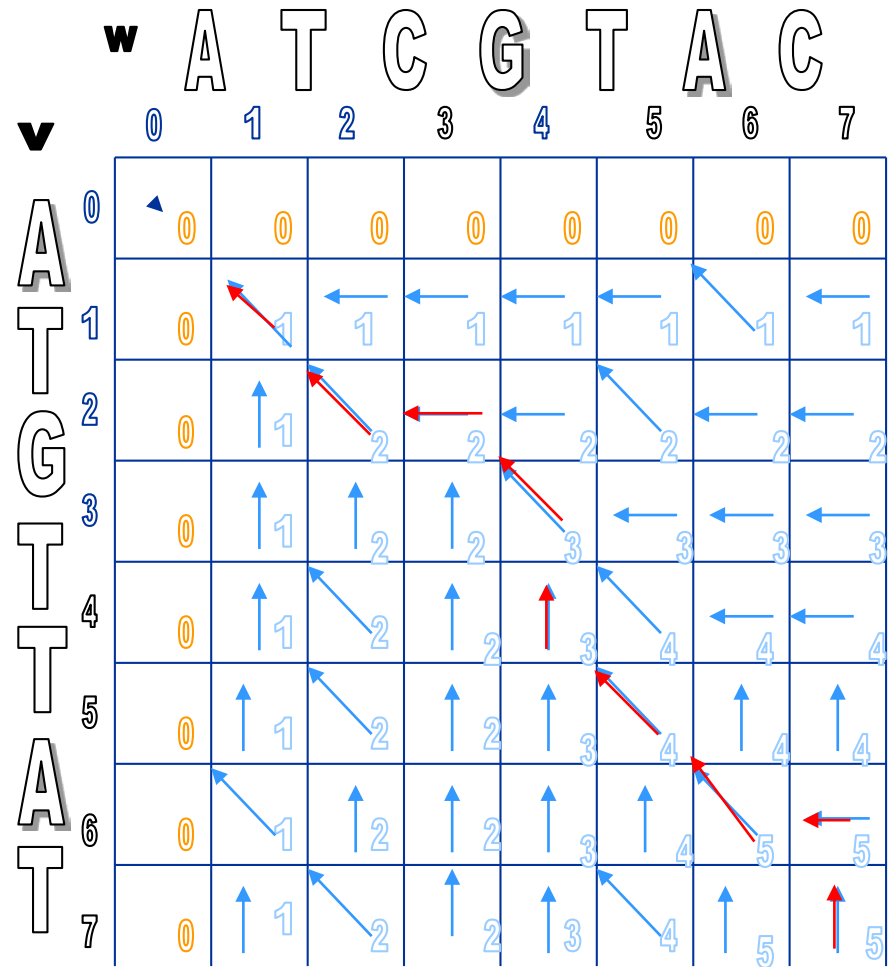
```

1. LCS(v, w)
2.   for  $i \leftarrow 1$  to  $n$ 
3.      $s_{i,0} \leftarrow 0$ 
4.   for  $j \leftarrow 1$  to  $m$ 
5.      $s_{0,j} \leftarrow 0$ 
6.   for  $i \leftarrow 1$  to  $n$ 
7.     for  $j \leftarrow 1$  to  $m$ 
8.        $s_{i,j} \leftarrow \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1, \text{ if } v_i = w_j \end{cases}$ 
9.        $b_{i,j} \leftarrow \begin{cases} \uparrow & \text{if } s_{i,j} = s_{i-1,j} \\ \leftarrow & \text{if } s_{i,j} = s_{i,j-1} \\ \nwarrow & \text{if } s_{i,j} = s_{i-1,j-1} + 1 \end{cases}$ 
10.      •
11.      •
12.      •
13.      •
14.      •
15.      •
16.      •
17.      •
18.      •
19.      •
20.      •
21.      •
22.      •
23.      •
24.      •
25.      •
26.      •
27.      •
28.      •
29.      •
30.      •
31.      •
32.      •
33.      •
34.      •
35.      •
36.      •
37.      •
38.      •
39.      •
40.      •
41.      •
42.      •
43.      •
44.      •
45.      •
46.      •
47.      •
48.      •
49.      •
50.      •
51.      •
52.      •
53.      •
54.      •
55.      •
56.      •
57.      •
58.      •
59.      •
60.      •
61.      •
62.      •
63.      •
64.      •
65.      •
66.      •
67.      •
68.      •
69.      •
70.      •
71.      •
72.      •
73.      •
74.      •
75.      •
76.      •
77.      •
78.      •
79.      •
80.      •
81.      •
82.      •
83.      •
84.      •
85.      •
86.      •
87.      •
88.      •
89.      •
90.      •
91.      •
92.      •
93.      •
94.      •
95.      •
96.      •
97.      •
98.      •
99.      •
100.     return ( $s_{n,m}$ ,  $b$ )

```

Now What?

- $\text{LCS}(v, w)$ created the alignment grid
- Now we need a way to read the best alignment of v and w
- Follow the arrows backwards from sink



Printing LCS: Backtracking

```
1. PrintLCS(b,v,i,j)
2.   if  $i = 0$  or  $j = 0$ 
3.     return
4.   if  $b_{i,j} = \nwarrow$ 
5.     PrintLCS(b,v,i-1,j-1)
6.     print  $v_i$ 
7.   else
8.     if  $b_{i,j} = \nearrow$ 
9.       PrintLCS(b,v,i-1,j)
10.    else
11.      PrintLCS(b,v,i,j-1)
```

LCS Runtime

- It takes $O(nm)$ time to fill in the $n \times m$ dynamic programming matrix.
- Why $O(nm)$? The pseudocode consists of a nested “for” loop inside of another “for” loop to set up a $n \times m$ matrix.

Why does DP work?

- Avoids re-computing the same sub-problems
- Limits the amount of work done in each step

When is DP applicable?

- **Optimal substructure:** Optimal solution to problem (instance) contains optimal solutions to sub-problems
- **Overlapping sub-problems:** Limited number of distinct sub-problems, repeated many many times

Next: Sequence Alignment

- Global Alignment
- Scoring Matrices
- Local Alignment
- Alignment with Affine Gap Penalties

From LCS to Alignment

- The Longest Common Subsequence (LCS) problem—the simplest form of sequence alignment – allows only insertions and deletions (no mismatches).
- In the LCS Problem, we scored 1 for matches and 0 for indels
- Consider penalizing indels and mismatches with negative scores
- Simplest *scoring schema*:
 - +1 : match premium
 - μ : mismatch penalty
 - σ : indel penalty

Simple Scoring

- When mismatches are penalized by $-\mu$, indels are penalized by $-\sigma$, and matches are rewarded with $+1$, the resulting score is:

$$\#matches - \mu(\#mismatches) - \sigma(\#indels)$$

The Global Alignment Problem

Find the best alignment between two strings under a given scoring schema

Input : Strings **v** and **w** and a scoring schema

Output : Alignment of maximum score

$$\begin{array}{l} \uparrow \rightarrow = -\sigma \\ \searrow \left\{ \begin{array}{l} = 1 \text{ if match} \\ = -\mu \text{ if mismatch} \end{array} \right. \end{array}$$

$$s_{i,j} = \max \left\{ \begin{array}{l} s_{i-1,j-1} + 1 \text{ if } v_i = w_j \\ s_{i-1,j-1} - \mu \text{ if } v_i \neq w_j \\ s_{i-1,j} - \sigma \\ s_{i,j-1} - \sigma \end{array} \right.$$

μ : mismatch penalty
 σ : indel penalty

Scoring Matrices

To generalize scoring, consider a $(4+1) \times (4+1)$ **scoring matrix** δ .

In the case of an amino acid sequence alignment, the scoring matrix would be a $(20+1) \times (20+1)$ size. The addition of 1 is to include the score for comparison of a gap character “-”.

This will simplify the algorithm as follows:

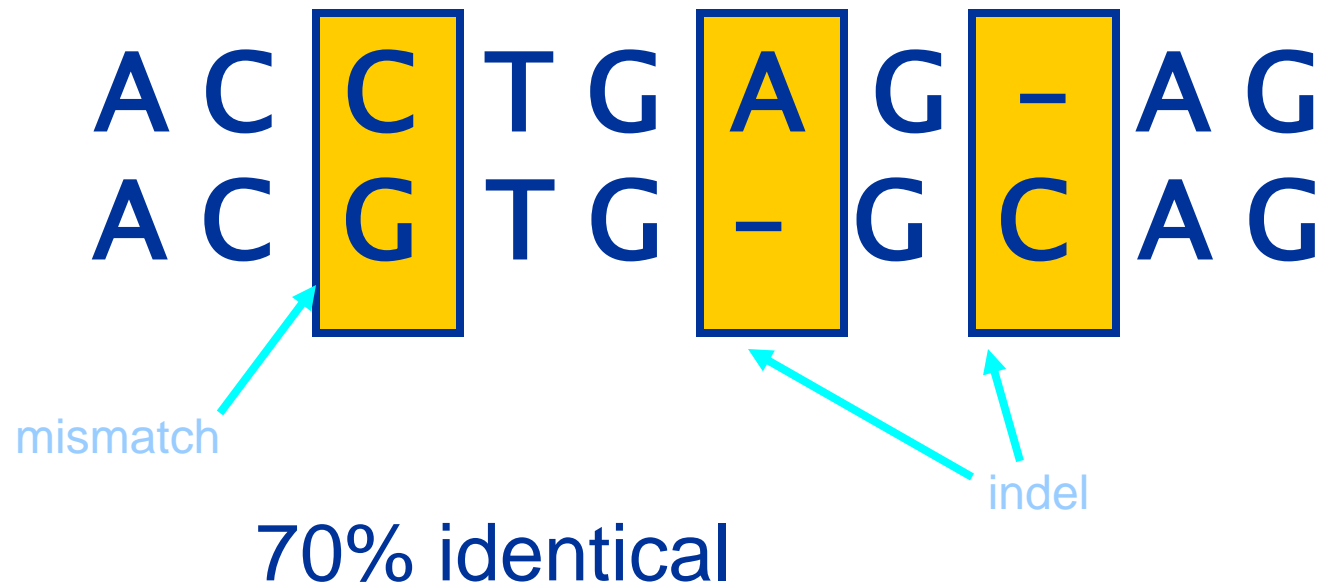
$$s_{i,j} = \max \left\{ \begin{array}{l} s_{i-1,j-1} + \delta(v_i, w_j) \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \end{array} \right.$$

Measuring Similarity

- Measuring the extent of similarity between two sequences
 - Based on percent sequence identity
 - Based on conservation

Percent Sequence Identity

- The extent to which two nucleotide or amino acid sequences are invariant



Making a Scoring Matrix

- Scoring matrices are created based on biological evidence.
- Alignments can be thought of as two sequences that differ due to mutations.
- Some of these mutations have little effect on the protein's function, therefore some penalties, $\delta(v_i, w_j)$, will be less harsh than others.

Scoring Matrix: Example

	A	R	N	K
A	5	-2	-1	-1
R	-	7	-1	3
N	-	-	7	0
K	-	-	-	6

AKRANR

KAAANK

$$-1 + (-1) + (-2) + 5 + 7 + 3 = 11$$

- Notice that although R and K are different amino acids, they have a positive score.
- Why? They are both positively charged amino acids → will not greatly change function of protein.

Conservation

- Amino acid changes that tend to preserve the physico-chemical properties of the original residue
 - Polar to polar
 - aspartate → glutamate
 - Nonpolar to nonpolar
 - alanine → valine
 - Similarly behaving residues
 - leucine to isoleucine

Scoring matrices

- Amino acid substitution matrices
 - PAM
 - BLOSUM
- DNA substitution matrices
 - DNA is less conserved than protein sequences
 - Less effective to compare coding regions at nucleotide level