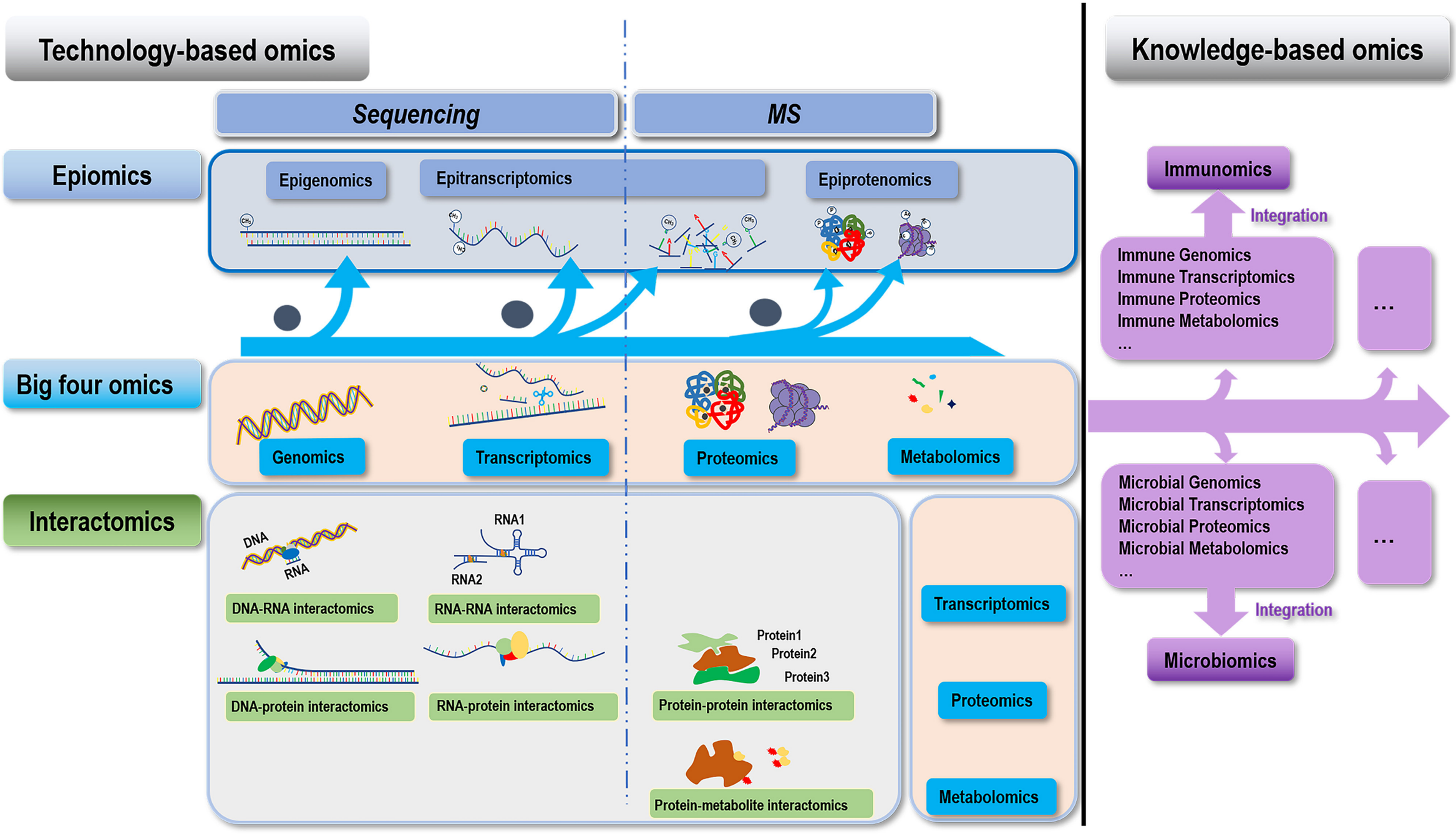
Bioinformatics

Lecture 1

Weidong Tian, School of Life Sciences, Fudan University

Sept 5, 2024

So many types of omics data



Pairwise sequence alignment

What is pairwise sequence alignment?

A protein sequence alignment

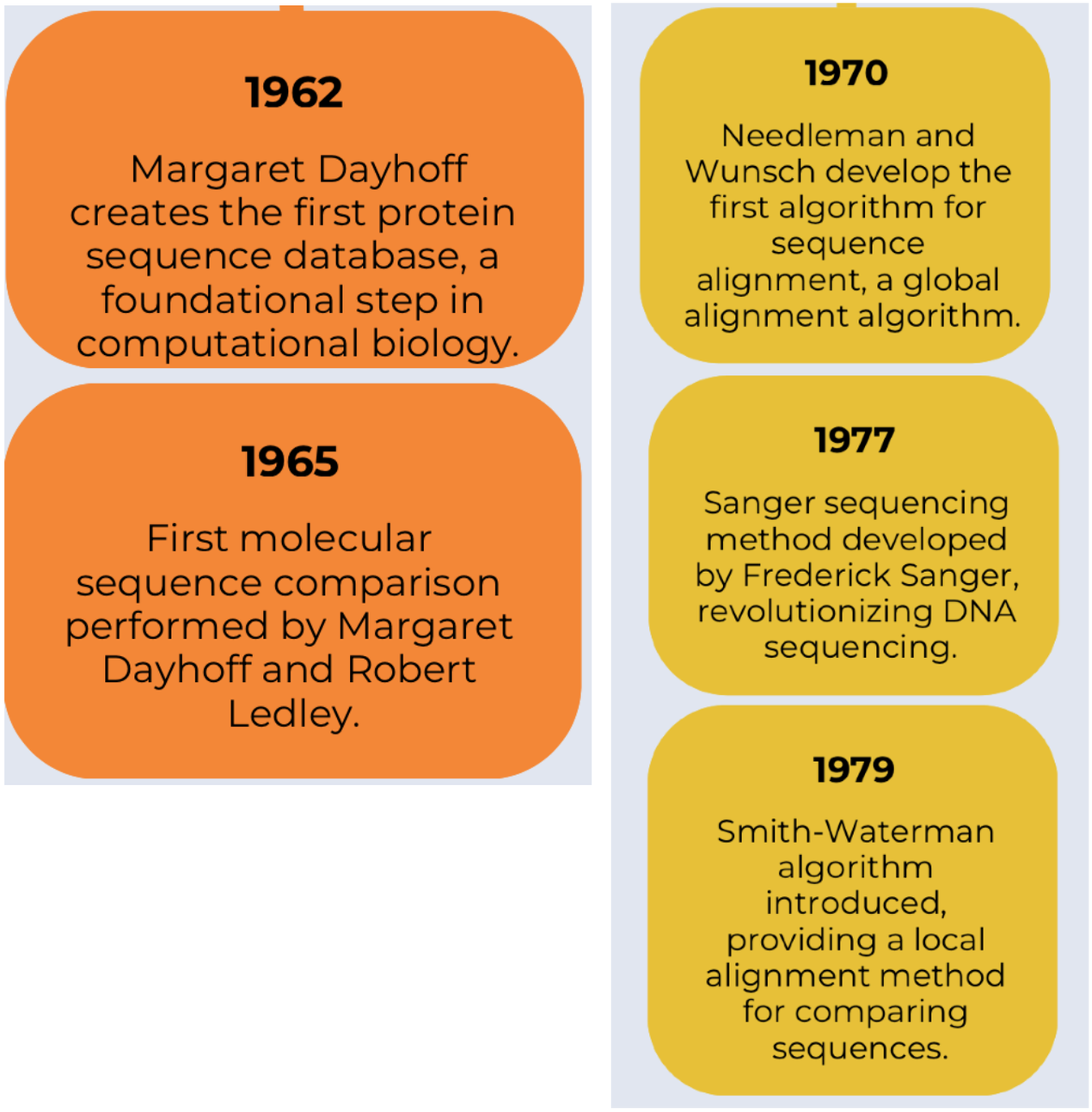
```
MSTGAVLIY--TSILIKECHAMPAGNE-----
---GGILLFHRTHELIKESHAMANDEGGSNNS
    *   *      *   **** ***
```

A DNA sequence alignment

```
attcgttggcaaatcgccctatccggccttaa
att---tggcggatcg-cctctacgggcc----
***    ****   **** **      ******
```

Pairwise sequence alignment is a computational technique used to mutually arrange two biological sequences (DNA, RNA, or protein) to identify regions of similarity or dissimilarity. This comparison can provide insights into evolutionary relationships, functional similarities, or structural relationships between the sequences.

Why is pairwise sequence alignment important?



- Pairwise sequence alignment is one of the earliest and most fundamental techniques in bioinformatics. It provides a basic framework for the comparative analysis of biological sequences and is a crucial tool for understanding relationships between sequences.

- Many early and modern bioinformatics algorithms are based on pairwise sequence alignment. Classic alignment algorithms, such as Needleman-Wunsch and Smith-Waterman, are still widely used today, laying the theoretical foundation for bioinformatics analysis.

- It is widely used in sequence analysis, evolutionary relationship inference, functional prediction, and more. The introduction and development of pairwise sequence alignment not only fueled the rise of bioinformatics but also provided indispensable technical support for modern biological research.

Homology detection by sequence comparison

- **Homology** refers to the similarity between biological sequences (DNA, RNA, or proteins) that is due to shared ancestry. When two sequences are homologous, it means they have evolved from a common ancestral sequence. Homology is a key concept in evolutionary biology and bioinformatics, used to infer evolutionary relationships and predict the functions of genes and proteins.

- **Orthology**: Homologous sequences that diverged due to speciation. Orthologs are typically found in different species and often retain the same function across species. For example, the gene for hemoglobin in humans and chimpanzees is orthologous.

- **Paralogy**: Homologous sequences that arose from gene duplication within the same organism. Paralogs can evolve new functions, even though they originated from the same ancestral gene. For instance, the human hemoglobin and myoglobin genes are paralogs.

- **Homology vs. Similarity**: It's important to note that homology is not the same as similarity. Similarity refers to the observable likeness between sequences, while homology is an evolutionary concept based on common ancestry. Sequences can be similar without being homologous, particularly if the similarity is due to convergent evolution or chance.

Molecular phylogeny inferred from sequence comparison

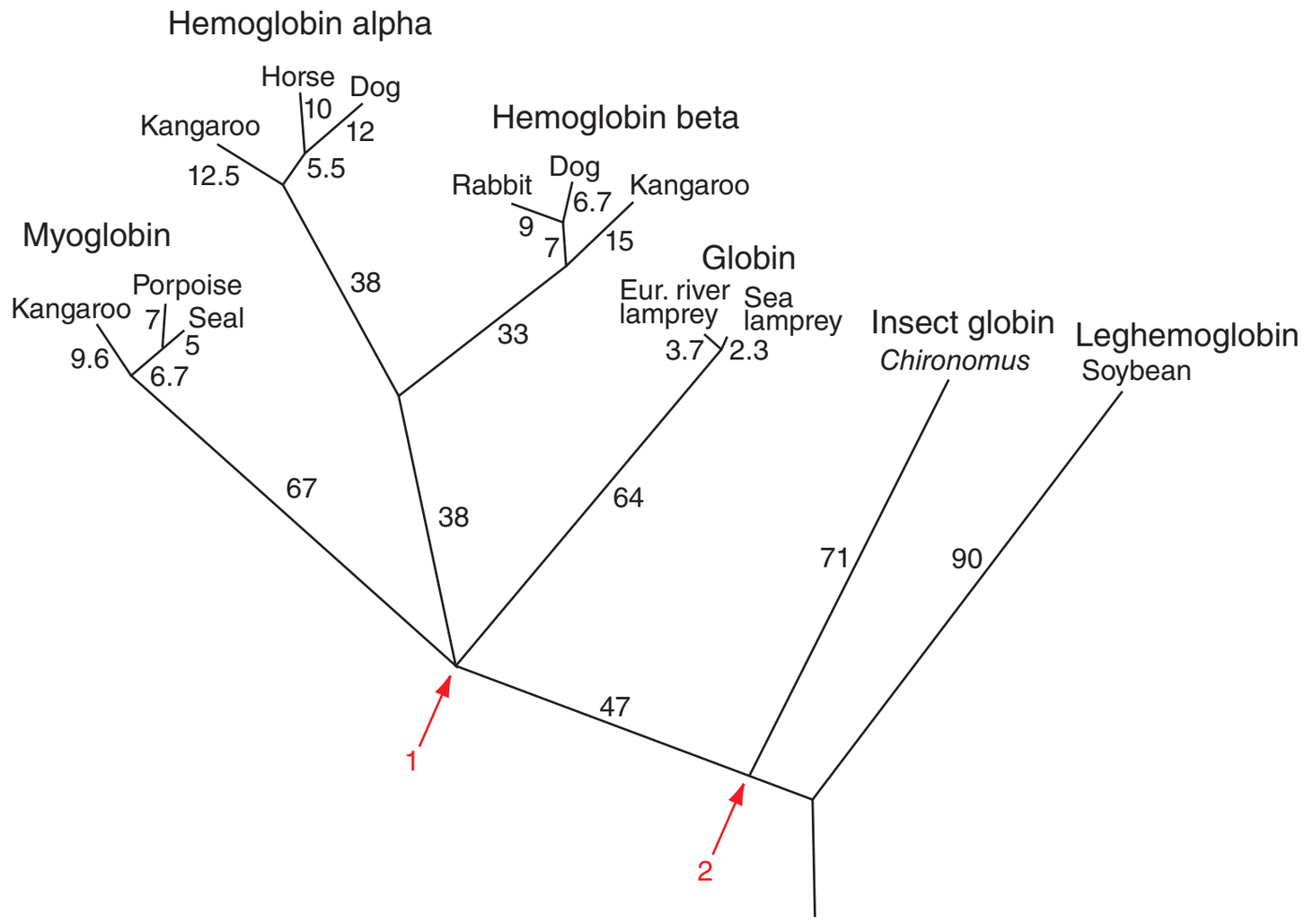


FIGURE 7.1 In the 1960s, several groups performed pioneering studies of globin phylogeny. This tree is modified from Dayhoff et al. (1972) who used maximum parsimony analysis to infer the relationships and history of 13 globins. The observed percent difference between sequences was corrected using the data on PAM matrices in **Table 3.3**. Arrow 1 indicates a node corresponding to the last common ancestor of the group of vertebrate globins, while arrow 2 indicates the ancestor of the insect and vertebrate globins (see text for details).

Source: Dayhoff et al. (1972). Reproduced with permission from National Biomedical Research Foundation.

Molecular phylogeny inferred from sequence comparison

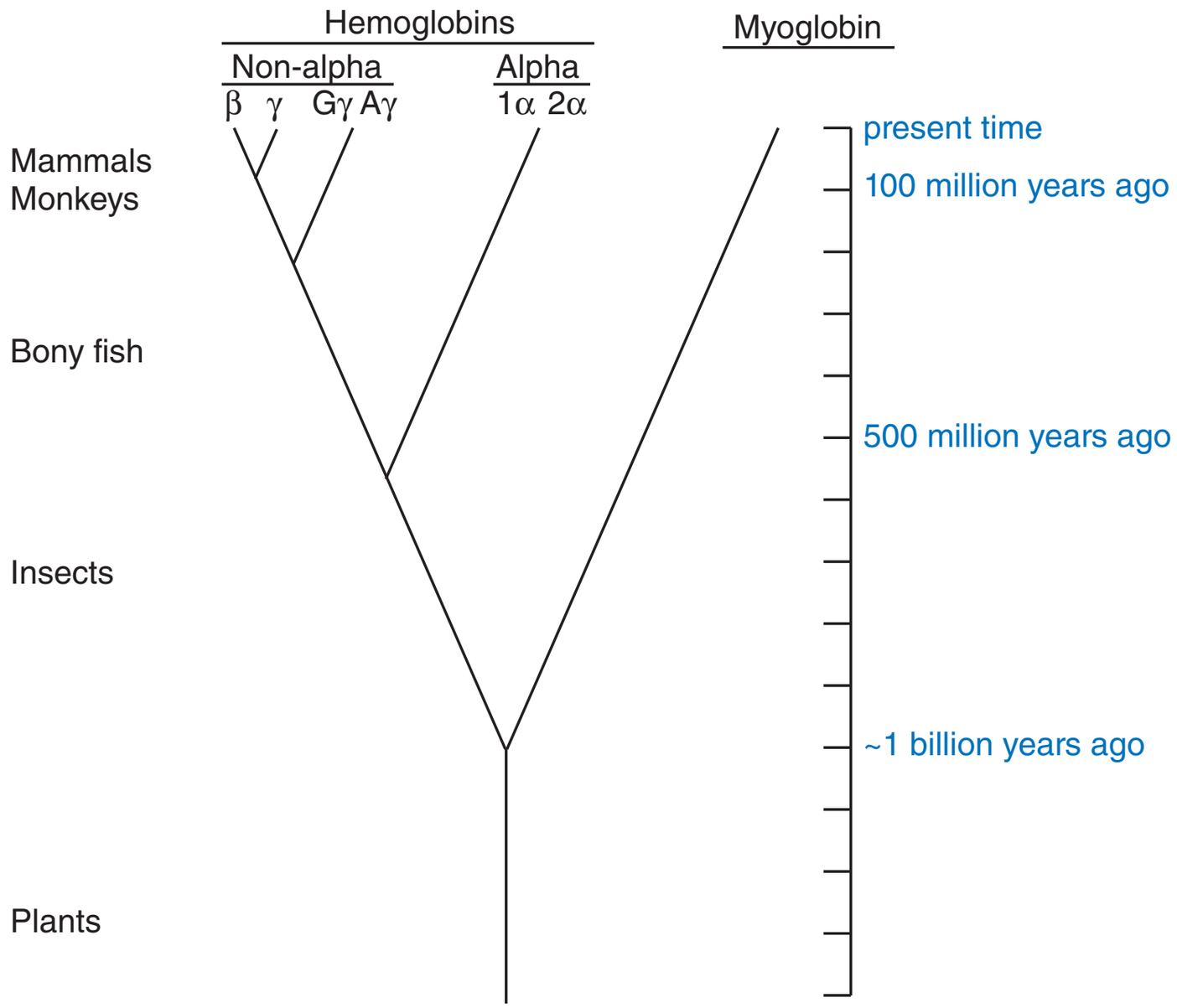


FIGURE 7.2 Dayhoff et al. (1972) summarized the relationship of the globin subfamilies in the context of evolutionary time. The dates of speciation events were inferred from fossil-based studies.

Source: Dayhoff et al. (1972). Reproduced with permission from National Biomedical Research Foundation.

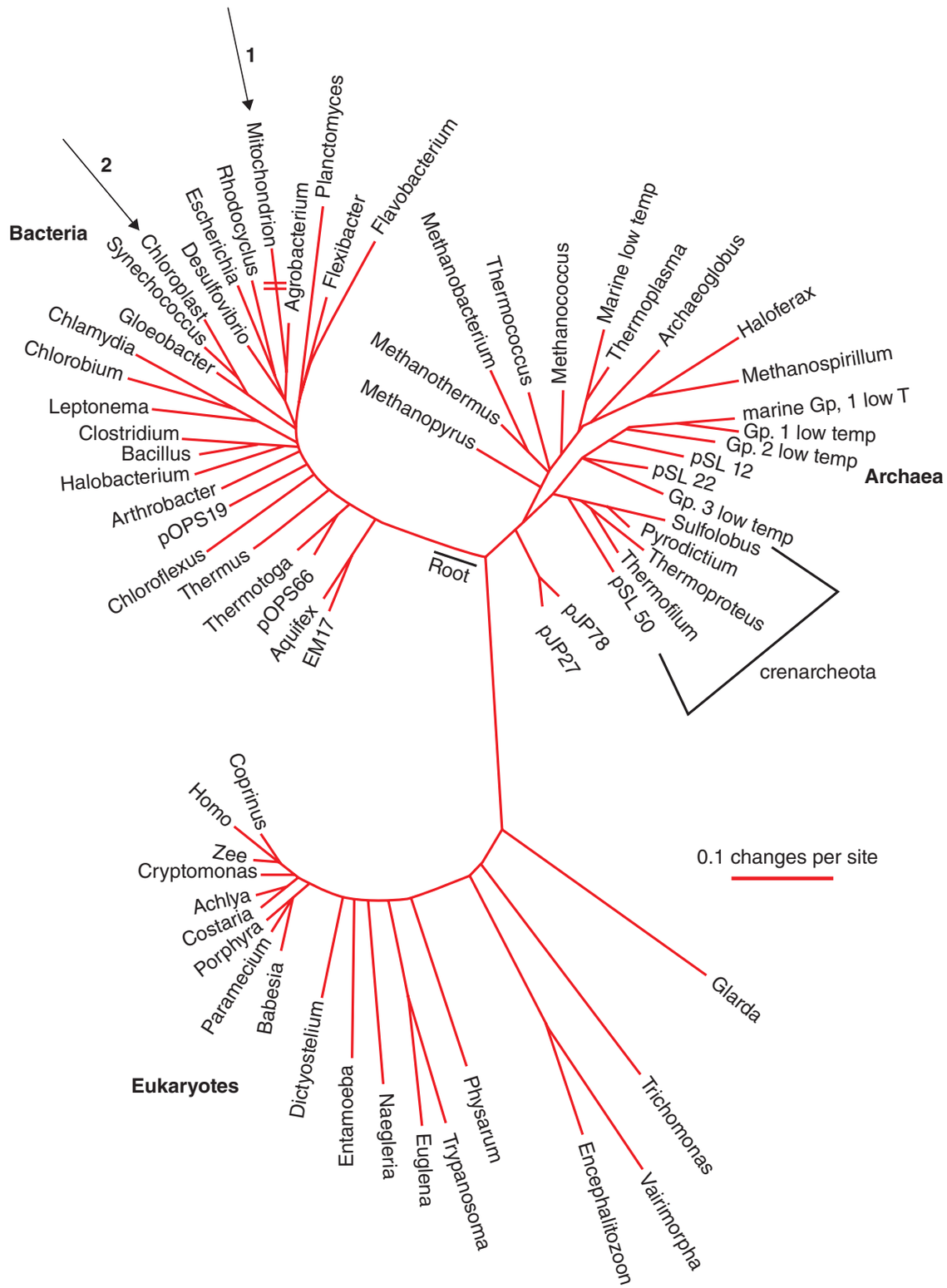
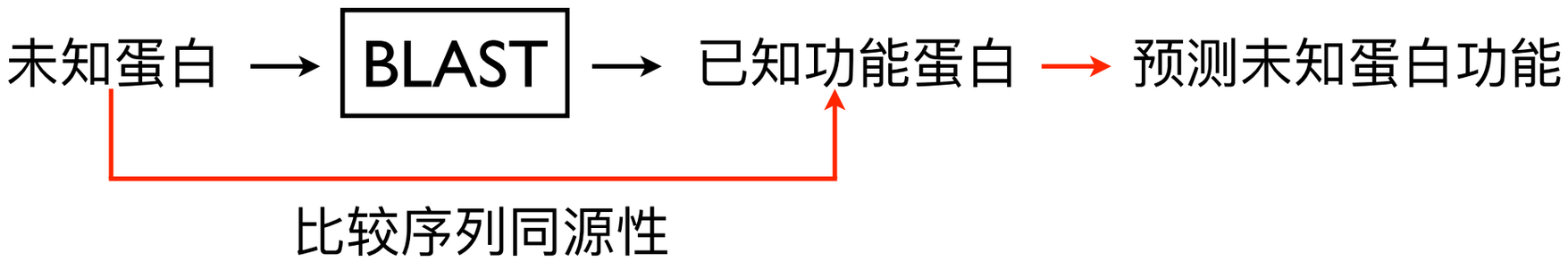


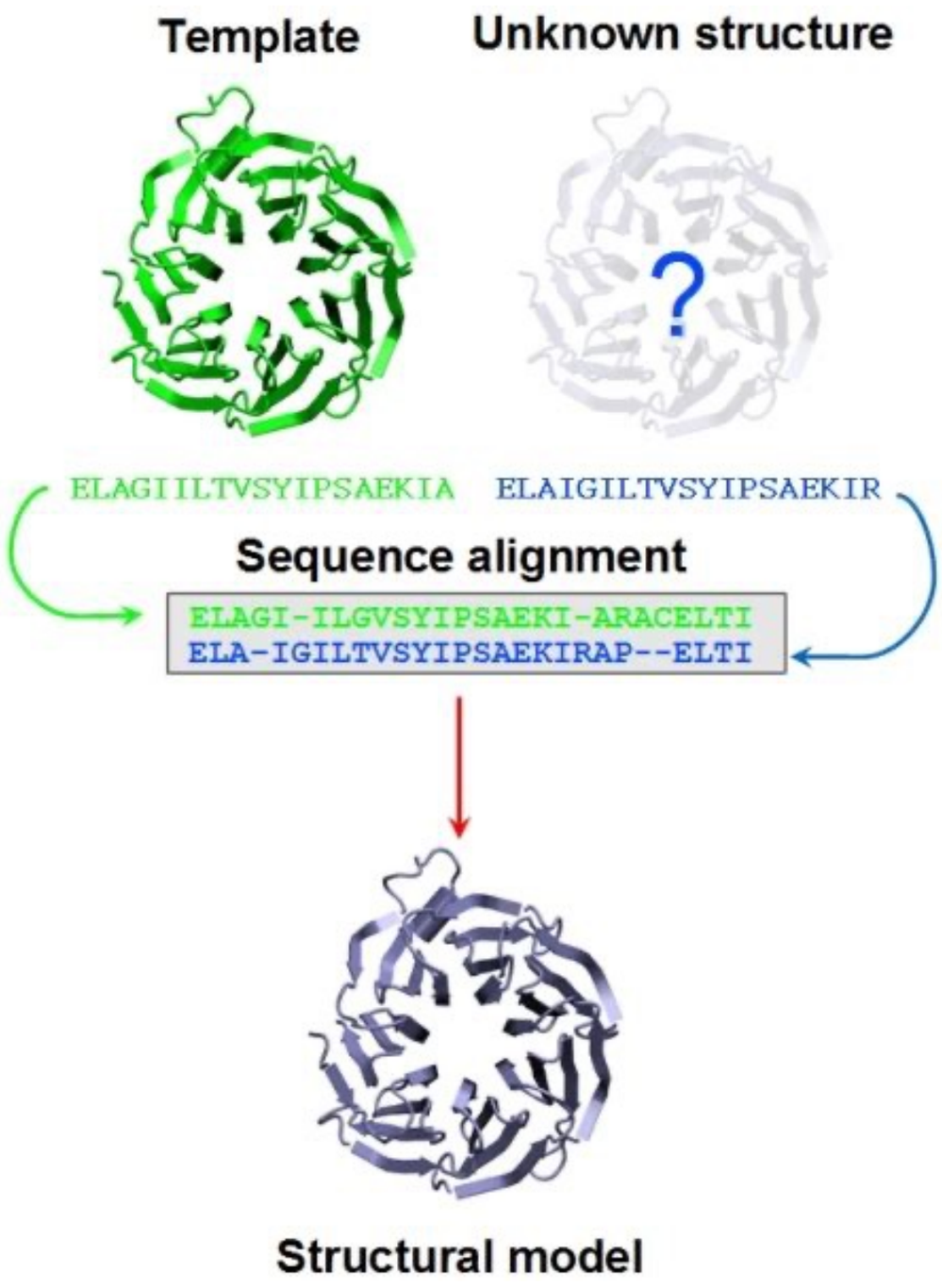
FIGURE 15.1 A global tree of life, based upon phylogenetic analysis of small-subunit rRNA sequences. Life is thought to have originated about 3.8 BYA in an anaerobic environment. The primordial life form (progenote) displayed the defining features of life (self-replication and evolution). The eukaryotic mitochondrion (arrow 1) and chloroplast (arrow 2) are indicated, showing their bacterial origins. Data from Barns *et al*. (1996), Hugenholtz and Pace (1996), and Pace (1997).

Function inference or genome annotation by sequence comparison

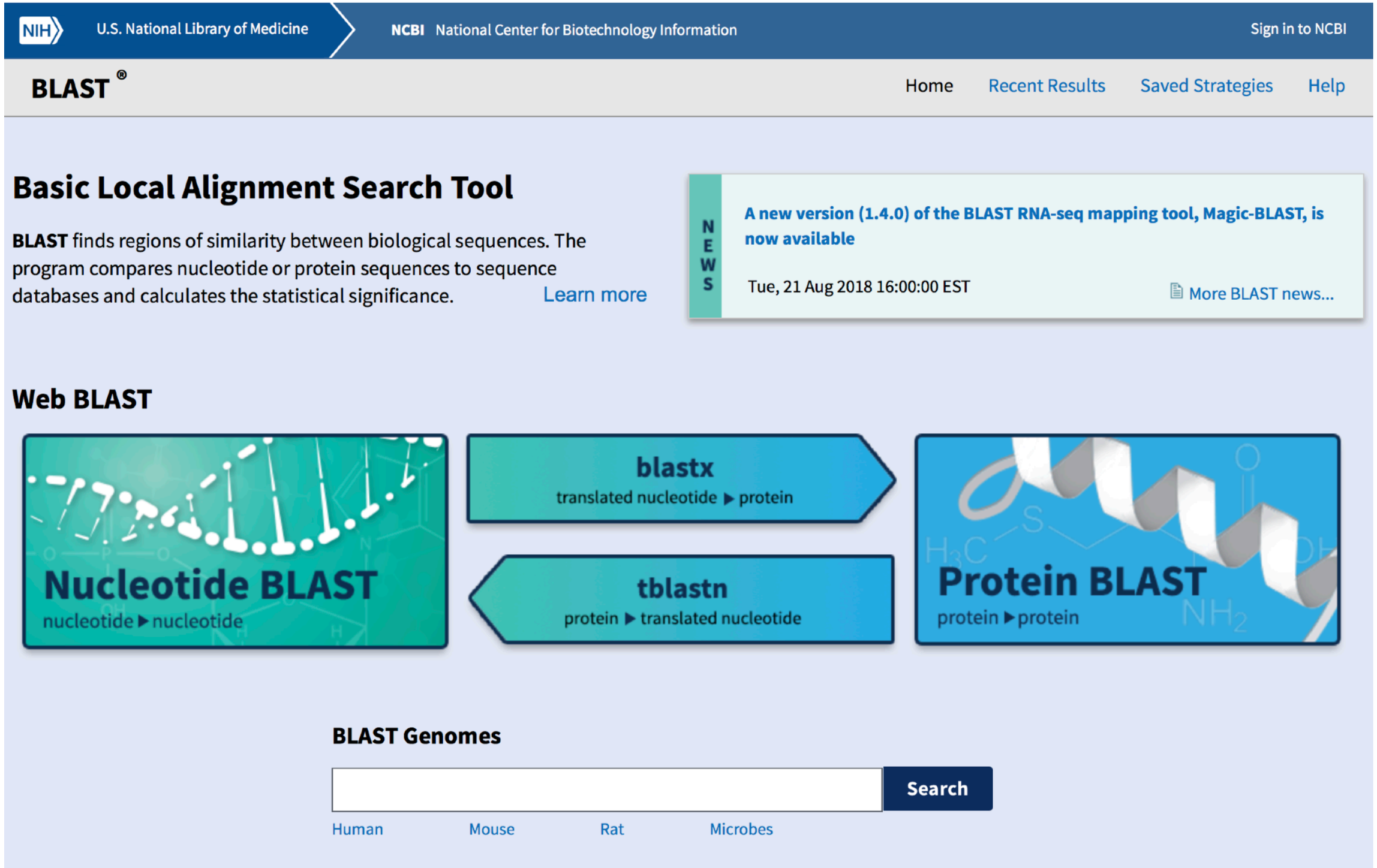


对酶功能相似性和序列同源性的研究表明，通过序列同源性预测蛋白功能相似性只有在序列同源性显著时才可靠。

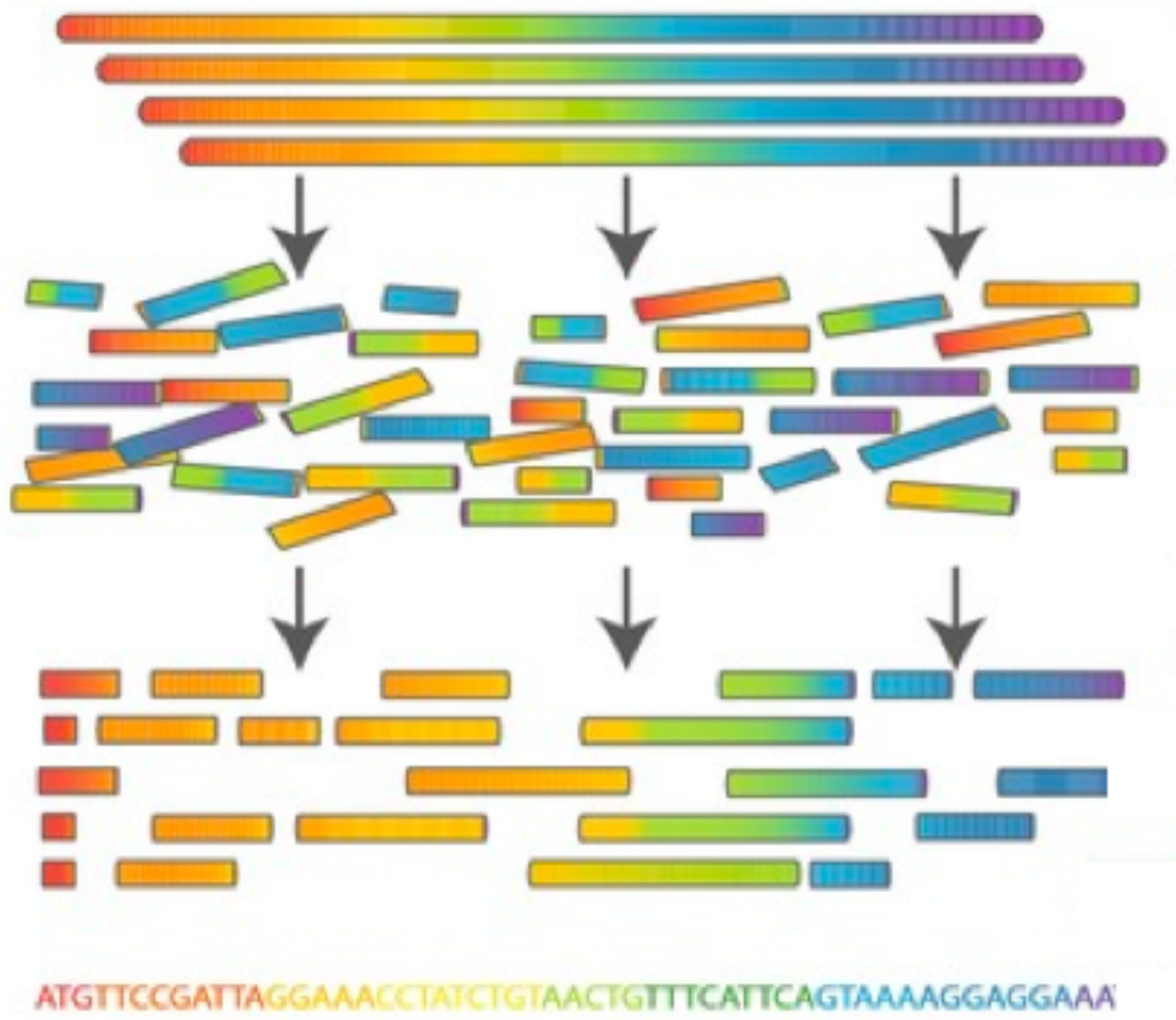
Structure inference from sequence comparison



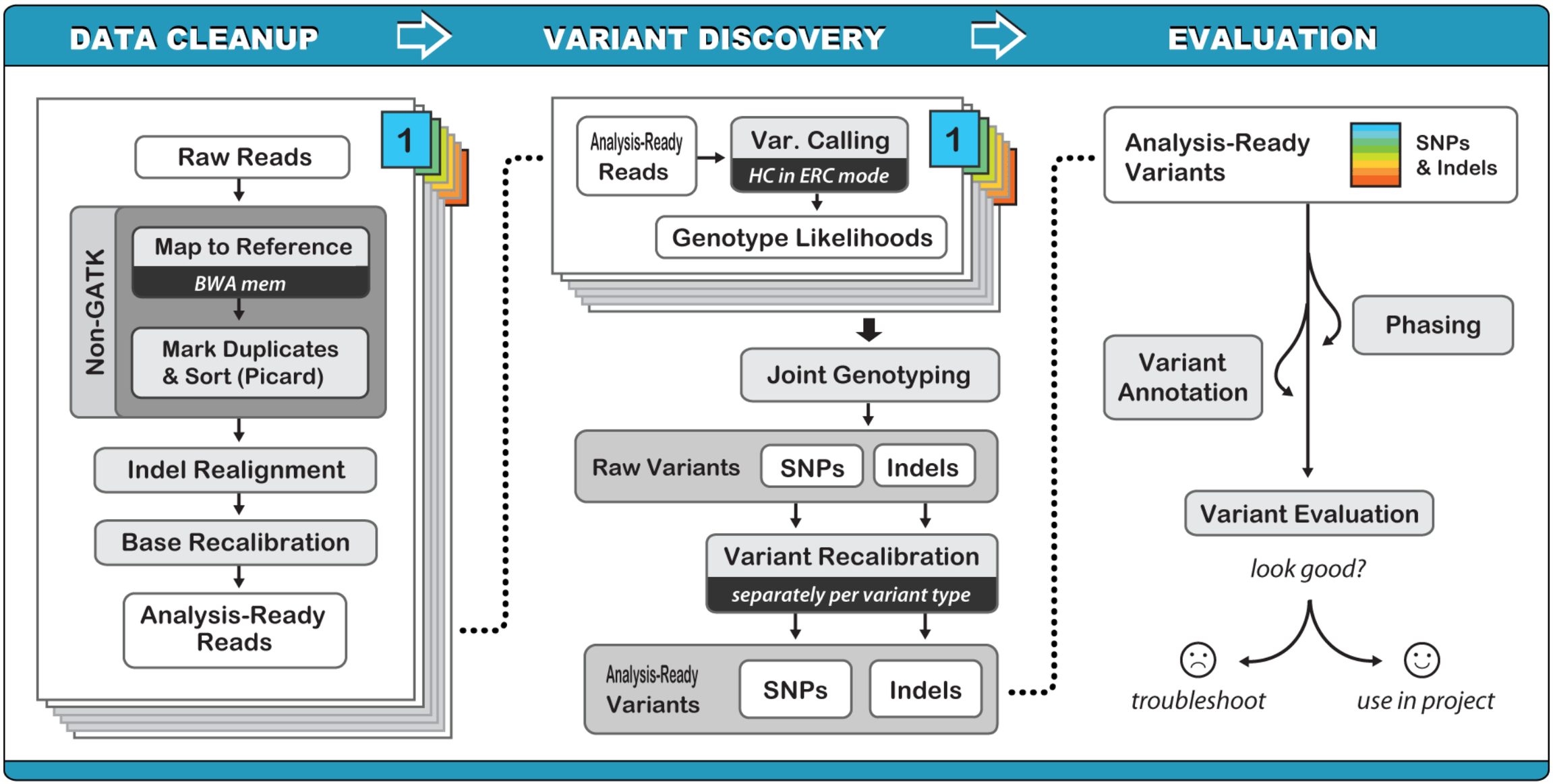
Database search by sequence comparison



Genome assembly from sequences comparison



Variant identification by sequence comparison



The pairwise sequence alignment problem

Input

Two biological sequences to be compared, usually of different or similar lengths.

Output

A protein sequence alignment

```
MSTGAVLIY--TSILIKECHAMPAGNE-----
---GGILLFHRTHELIKESHAMANDEGGSNNS
    *   *     *   **** ***
```

A DNA sequence alignment

```
attcgttggcaaatcgccctatccggccttaa
att---tggcggatcg-cctctacgggcc----
***   ****  **** **    ******
```

- **Goal:** The optimal alignment between two biological sequences to maximize their similarity according to a defined scoring system.

Aligning two DNA sequences

Given 2 DNA sequences **v** and **w**:

v: ATGTTAT m = 7

w: ATCGTAC n=7

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

letters of ***v***	A	T	--	G	T	T	A	T	--
letters of ***w***	A	T	C	G	T	--	A	--	C

5 matches

2 deletions

2 insertions

Goal: maximizes the number of matches and minimizes the number of mismatches and gaps (insertions and deletions).

Aligning two DNA sequences

incorporating indices

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

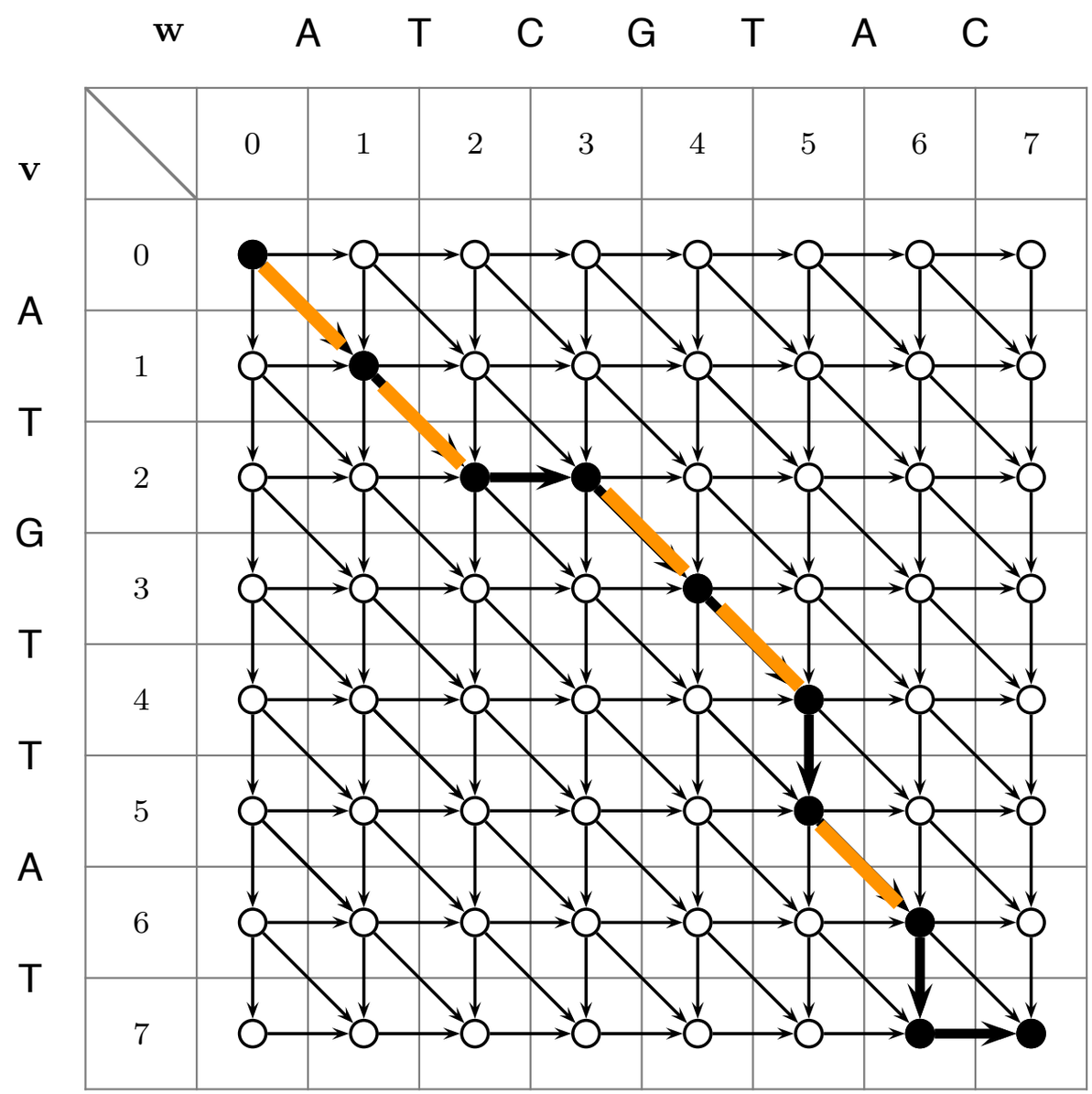
$$
\begin{array}{cccccccccc}
 & A & T & - & G & T & T & A & T & - \\
\begin{pmatrix} 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 1 \\ 1 \end{pmatrix} & \begin{pmatrix} 2 \\ 2 \end{pmatrix} & \begin{pmatrix} 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 3 \\ 4 \end{pmatrix} & \begin{pmatrix} 4 \\ 5 \end{pmatrix} & \begin{pmatrix} 5 \\ 5 \end{pmatrix} & \begin{pmatrix} 6 \\ 6 \end{pmatrix} & \begin{pmatrix} 7 \\ 6 \end{pmatrix} & \begin{pmatrix} 7 \\ 7 \end{pmatrix} \\
 & A & T & G & C & T & - & A & - & C
\end{array}
$$

Aligning two DNA sequences

$$\begin{pmatrix} & \mathsf{A} & \mathsf{T} & - & \mathsf{G} & \mathsf{T} & \mathsf{T} & \mathsf{A} & \mathsf{T} & - \\ 0 & 1 & 2 & 2 & 3 & 4 & 5 & 6 & 7 & 7 \\ 0 & 1 & 2 & 3 & 4 & 5 & 5 & 6 & 6 & 7 \\ & \mathsf{A} & \mathsf{T} & \mathsf{G} & \mathsf{C} & \mathsf{T} & - & \mathsf{A} & - & \mathsf{C} \end{pmatrix}$$

Showing the alignment on a two-dimensional grid

matches



$$\begin{array}{cccccccccc} \searrow & \searrow & \rightarrow & \searrow & \searrow & \downarrow & \searrow & \downarrow & \rightarrow \\ \mathsf{A} & \mathsf{T} & - & \mathsf{G} & \mathsf{T} & \mathsf{T} & \mathsf{A} & \mathsf{T} & - \\ \mathsf{A} & \mathsf{T} & \mathsf{C} & \mathsf{G} & \mathsf{T} & - & \mathsf{A} & - & \mathsf{C} \end{array}$$

The longest common subsequence problem

Longest Common Subsequence Problem:
Find the longest subsequence common to two strings.

Input: Two strings, **v** and **w**.

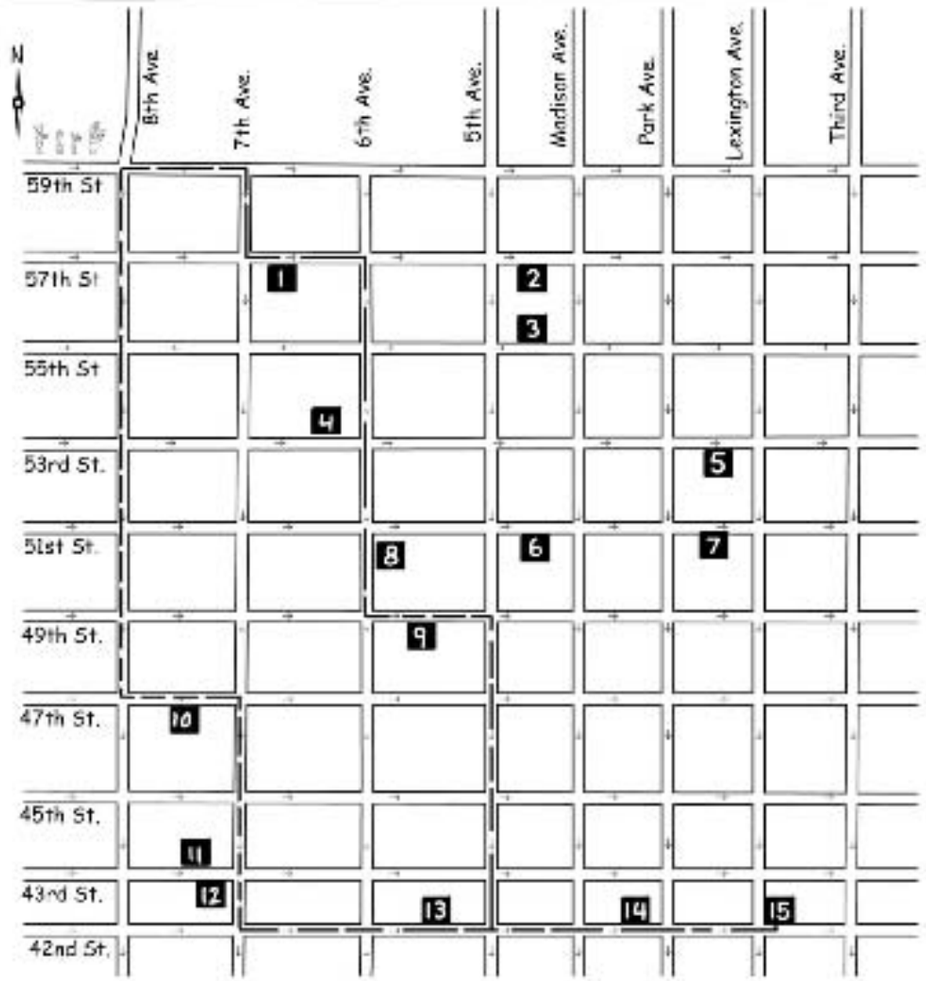
Output: The longest common subsequence of **v** and **w**.

- The LCS problem specifically looks for the longest subsequence that appears in both sequences in the same order, though not necessarily consecutively.

- If we consider only to maximize the number of matches, then the pairwise sequence alignment problem can be simplified to the LCS problem, which is easier to solve.

- In practice, the pairwise sequence alignment problem considers not only matches, but also mismatches and gaps.

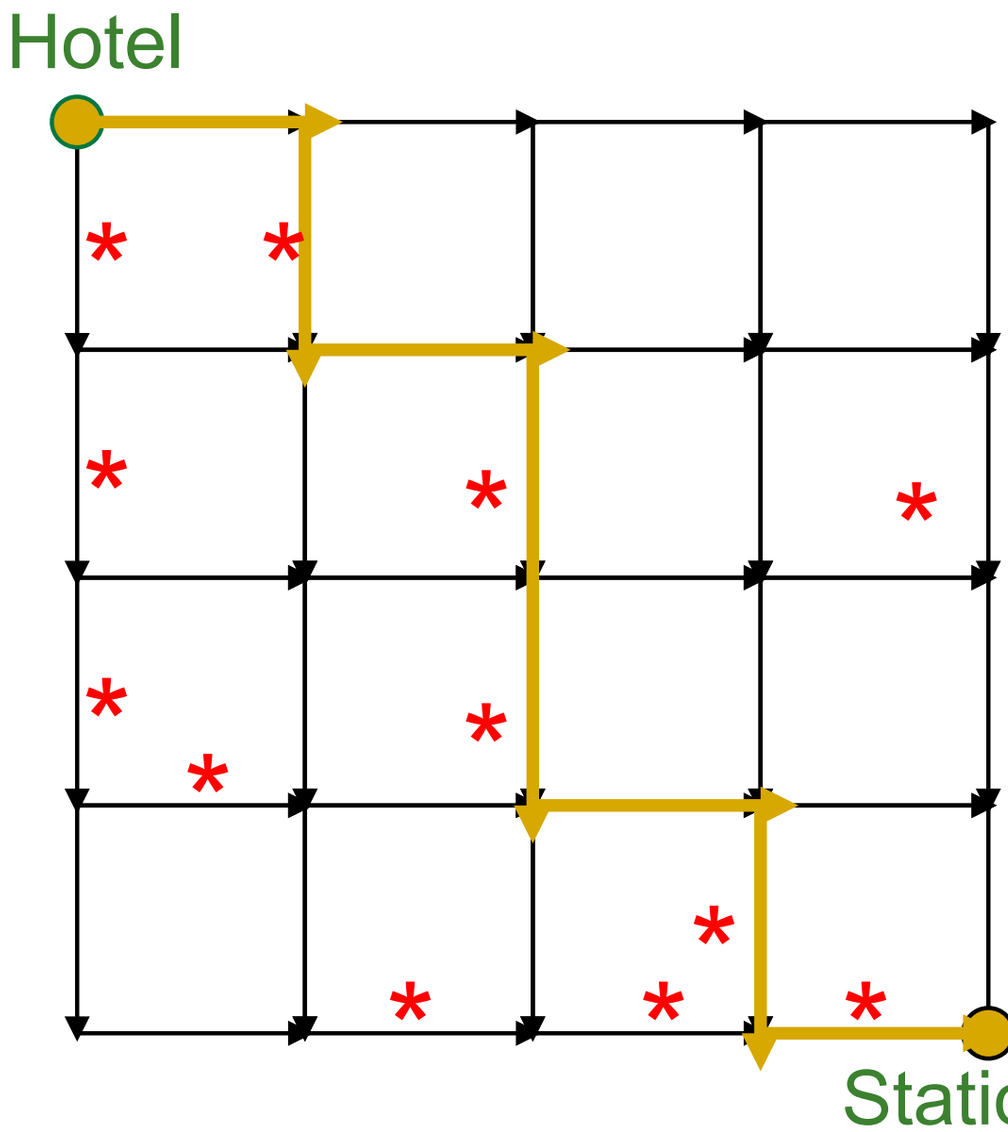
The Manhattan Tourist Problem

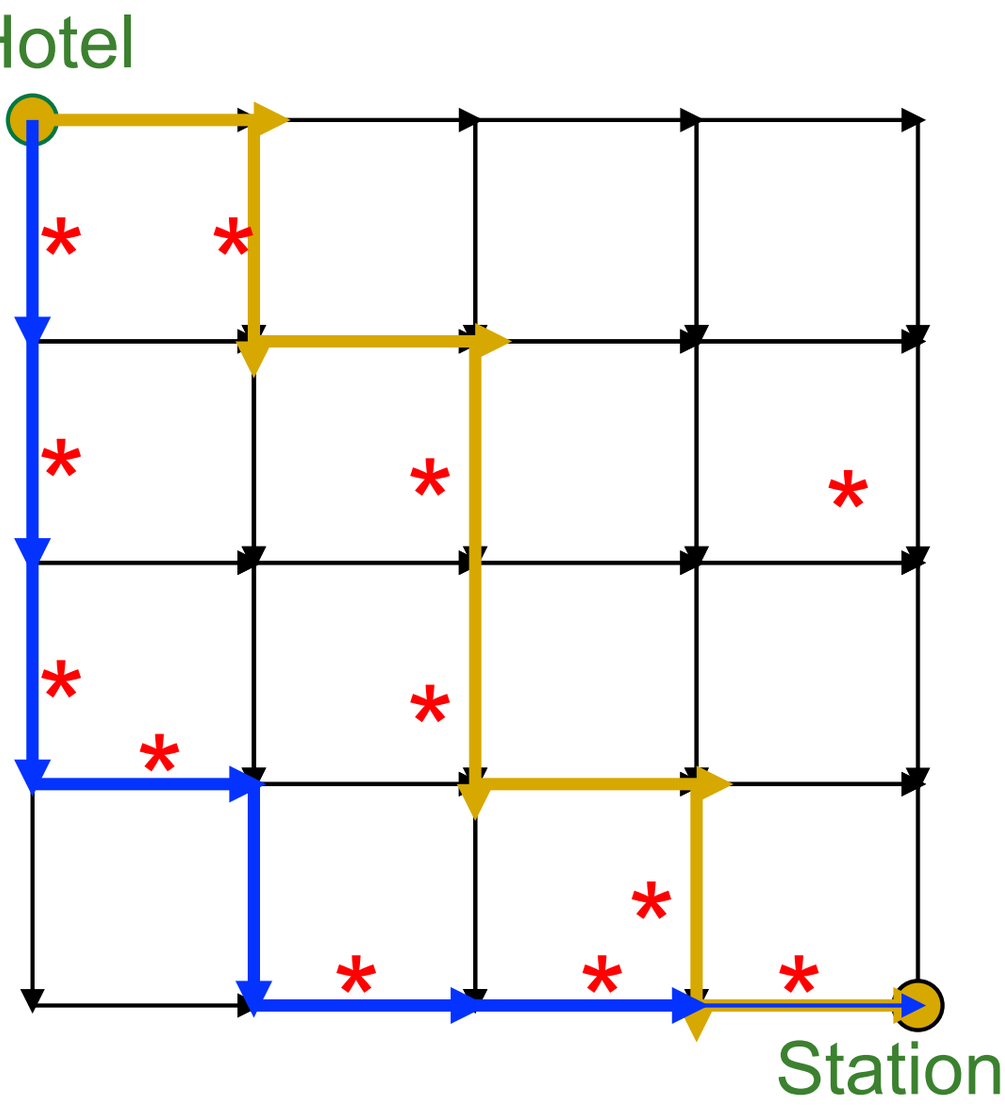


- Imagine that you are a tourist in Manhattan, whose streets are represented by the grid on the right.
- You are leaving town, and you want to see as many attractions (represented by *) as possible.
- Your time is limited: you only have time to travel east and south.
- What is the best path through town?

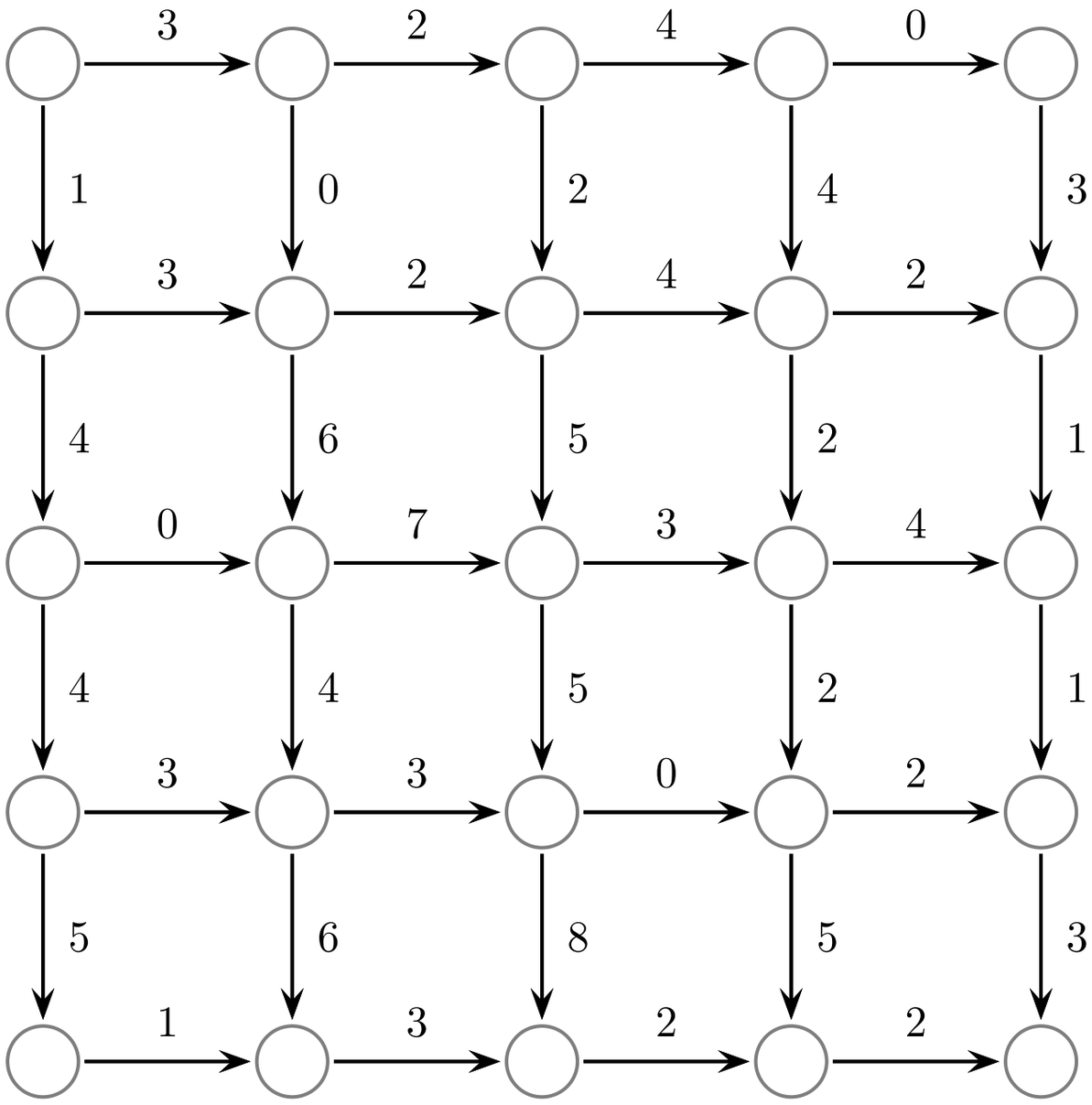
The Manhattan Tourist Problem

- What is the best path through town?





The Manhattan Tourist Problem



Manhattan Tourist Problem:
Find a longest path in a weighted grid.

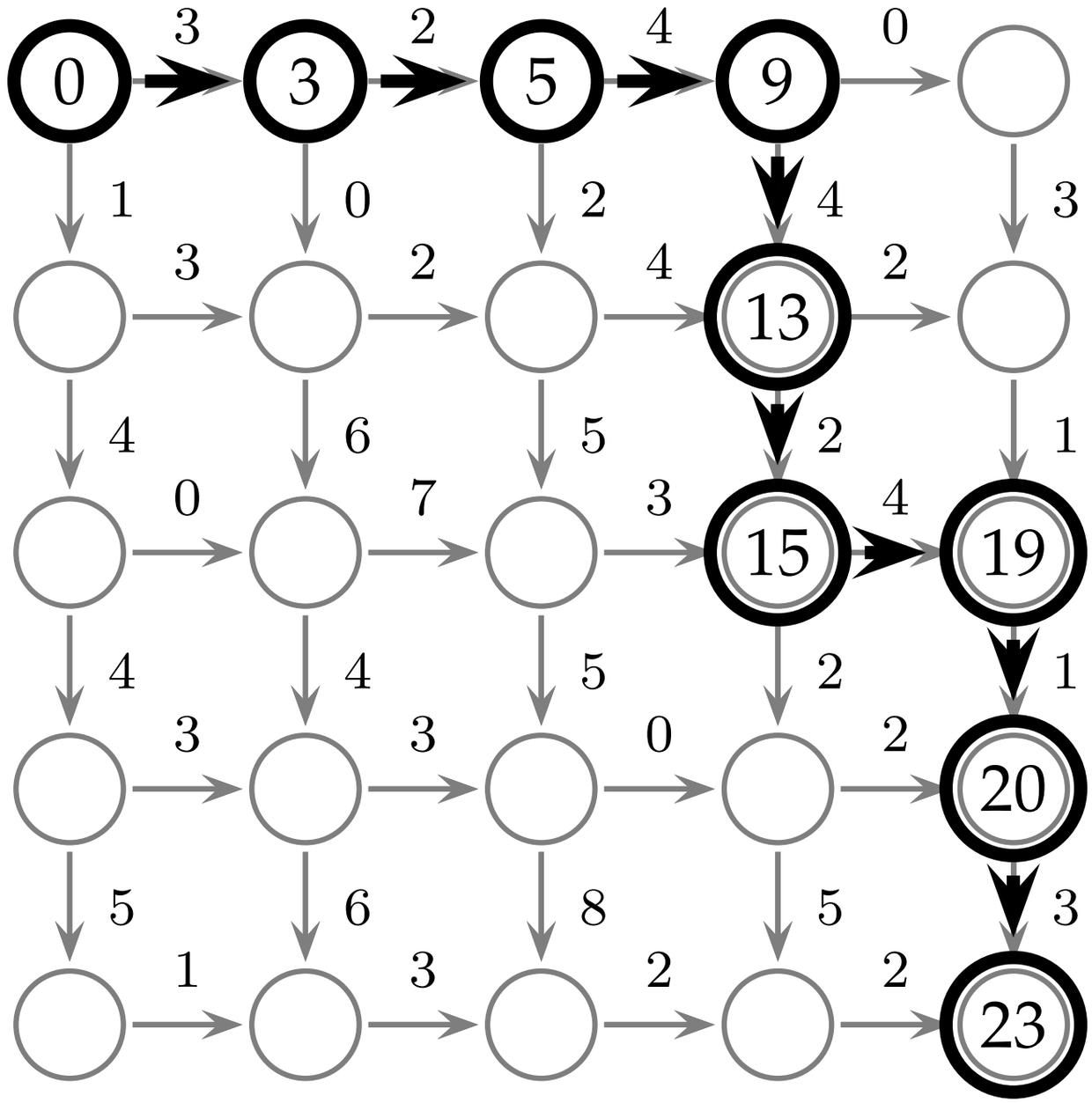
Input: A weighted grid G with two distinguished vertices: a *source* and a *sink*.

Output: A longest path in G from *source* to *sink*.

The Manhattan Tourist Problem— a greedy solution

- A greedy strategy would be to choose between two possible directions (south or east) by comparing how many attractions tourists would see if they moved one block south instead of moving one block east.

- This greedy strategy may provide rewarding sightseeing experience in the beginning but, a few blocks later, may bring you to an area of Manhattan you really do not want to be in.



The limitation of the greedy approach

- **Local vs. Global Optimization:** The greedy approach only considers the immediate best choice and does not account for the long-term consequences. This can lead to suboptimal paths, especially in cases where a slightly less optimal choice in the short term could lead to a significantly better outcome later.

- **Non-Optimality:** Since the MTP involves the accumulation of weights along a path, the greedy algorithm may miss the globally optimal solution, as it doesn’t backtrack or explore all possible paths.

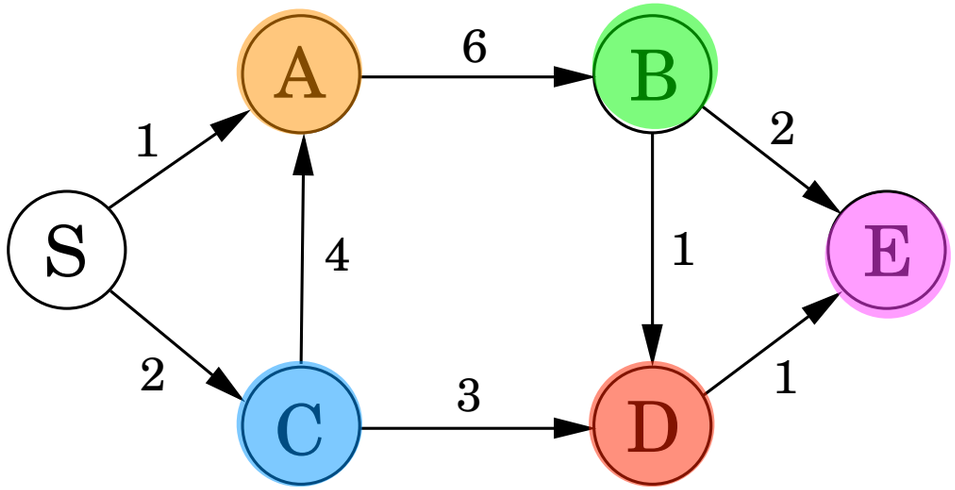
Solution to global optimization —the Dynamic Programming approach

- **Dynamic Programming (DP)** is a method used in algorithm design to solve complex problems by breaking them down into simpler subproblems. It is particularly effective for problems that exhibit two key properties: **overlapping subproblems** and **optimal substructure**.

- **Overlapping Subproblems:** The problem can be divided into subproblems that recur multiple times. Instead of solving these subproblems repeatedly, DP solves each subproblem once and stores its solution in a table (often called a memoization table) to avoid redundant computations.

- **Optimal Substructure:** The optimal solution to the problem can be constructed from the optimal solutions of its subproblems. This means that solving a problem optimally involves combining the optimal solutions of its smaller, constituent problems.

The shortest path problem



$$S \to A$$
$$S \to C \to A$$

$$S \to A \to B$$
$$S \to C \to A \to B$$

$$S \to C$$

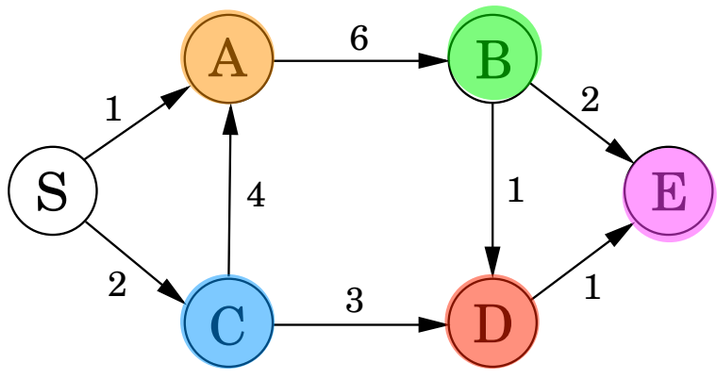
$$S \to C \to D$$
$$S \to C \to A \to B \to D$$
$$S \to A \to B \to D$$

$$S \to C \to A \to B \to E$$
$$S \to C \to A \to B \to D \to E$$
$$S \to C \to D \to E$$

Overlapping Subproblems:
S->A, S->B, S->C, S->D is calculated repeatedly; With DP, they are memorized in a table to save computation.

Optimal Substructure:
The optimal solution of S->E can be constructed from the optimal solutions of S->D and S->B.

Overlapping Subproblems and Optimal Substructure



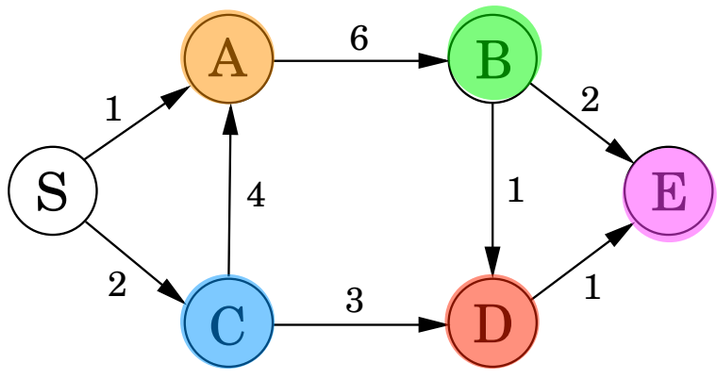
- Overlapping subproblems occur when a problem can be broken down into subproblems that are reused multiple times in the process of solving the original problem.

- Recognizing overlapping subproblems allows for the application of memoization or tabulation, reducing the time complexity from exponential to polynomial in many cases.

- Optimal substructure means that the optimal solution to a problem can be constructed from the optimal solutions of its subproblems.

- The optimal solution to the original problem is derived by combining the optimal solutions of the subproblems.

Overlapping Subproblems and Optimal Substructure



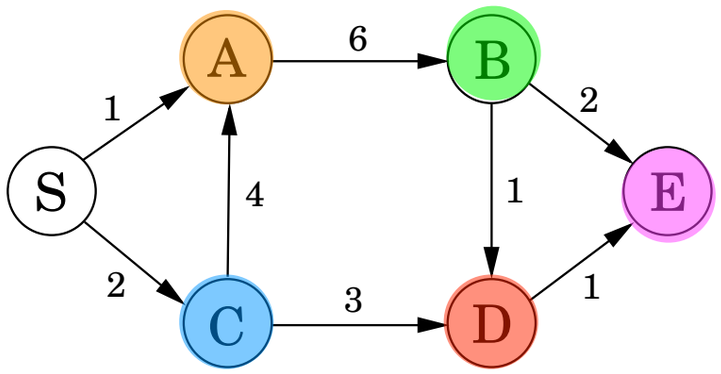
- Overlapping subproblems occur when a problem can be broken down into subproblems that are reused multiple times in the process of solving the original problem.

- Recognizing overlapping subproblems allows for the application of memoization or tabulation, reducing the time complexity from exponential to polynomial in many cases.

- Optimal substructure means that the optimal solution to a problem can be constructed from the optimal solutions of its subproblems.

- The optimal solution to the original problem is derived by combining the optimal solutions of the subproblems.

Overlapping Subproblems and Optimal Substructure



so to find the shortest path to D, we need only compare these two routes:

$$\texttt{dist}(D) = \min\{\texttt{dist}(B) + 1, \texttt{dist}(C) + 3\}.$$

A similar relation can be written for every node.

```
initialize all dist(·) values to ∞
dist(s) = 0
for each v ∈ V\{s}, in linearized order:
    dist(v) = min_{(u,v)∈E} {dist(u) + l(u, v)}
```

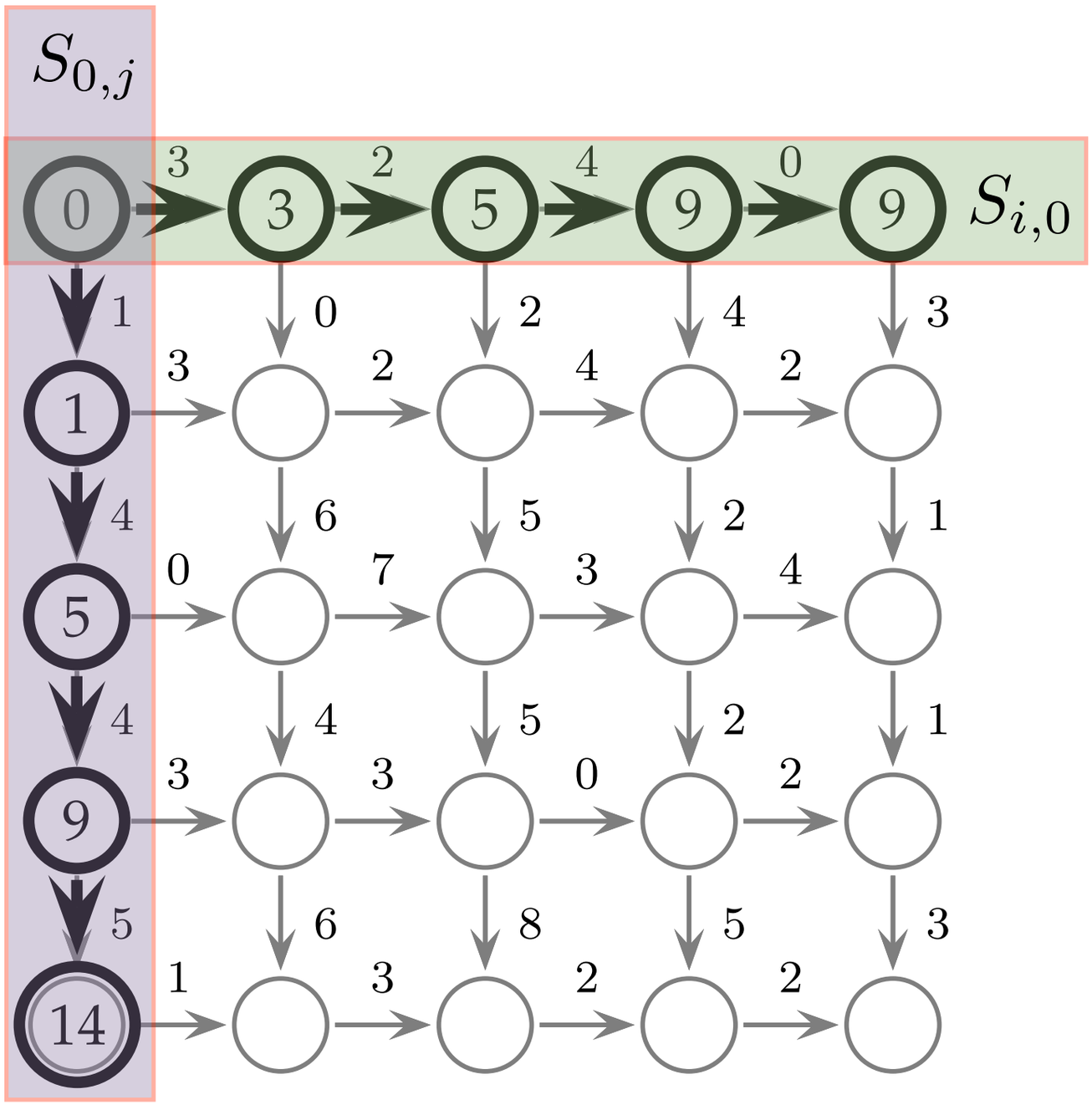
Dynamic Programming

- Richard Bellman introduced dynamic programming as a method to solve complex decision-making problems by breaking them down into simpler subproblems. He used this technique to address issues in operations research and control theory, particularly where decisions need to be made sequentially over time.

- The term "programming" in this context is related to "mathematical programming," which is an optimization technique. It was used to describe the process of finding optimal solutions to problems defined by certain constraints and objective functions.

- Dynamic programming is called "dynamic programming" because it involves a dynamic approach to solving problems by breaking them down into simpler, overlapping subproblems and systematically solving these subproblems to build up the solution to the overall problem. The term reflects the evolving nature of problem-solving and the systematic, iterative approach used to address optimization and decision-making problems.

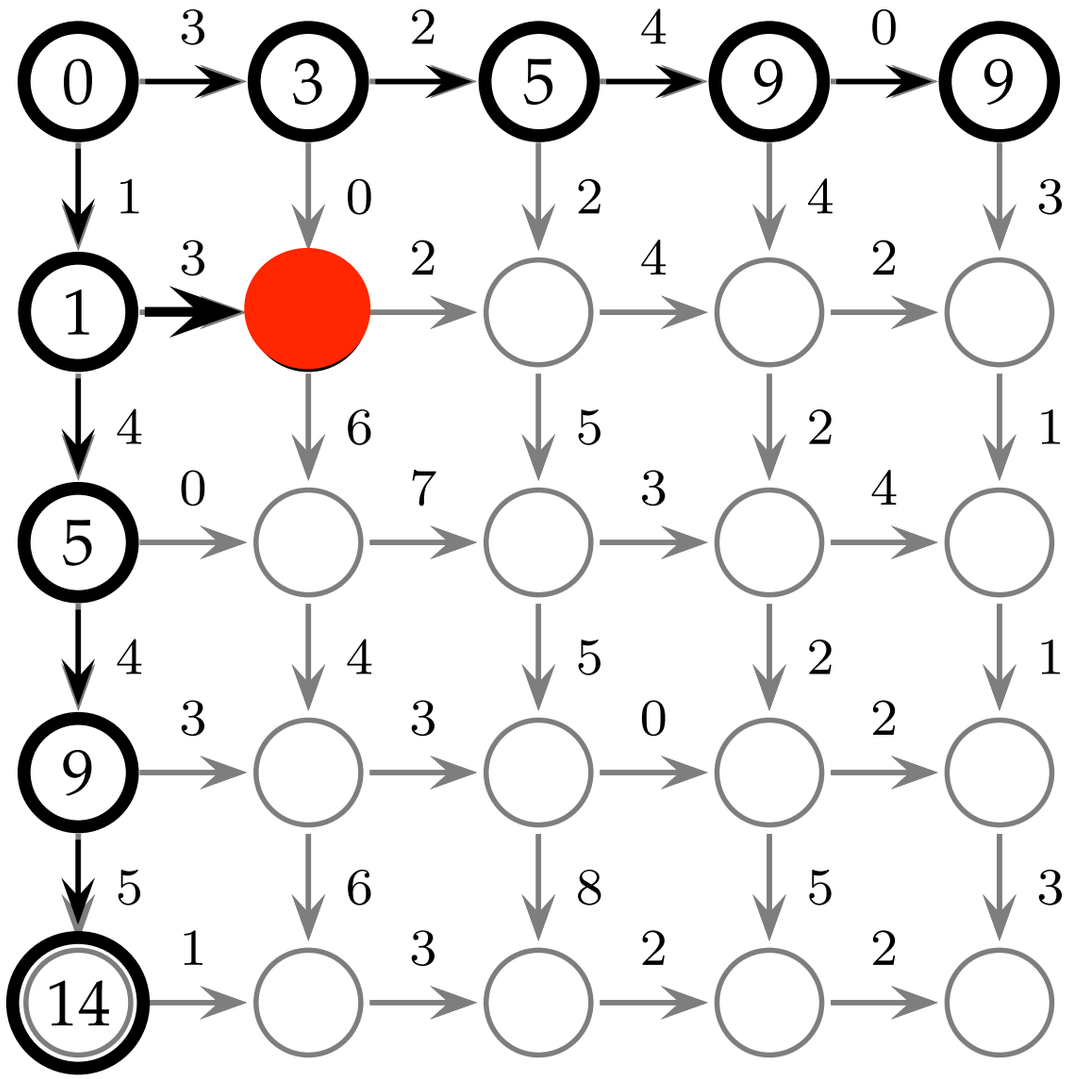
The Manhattan Tourist Problem — the dynamic programming solution



- Instead of solving the MTP problem directly, that is, finding the longest path from source (0, 0) to sink (n, m), we solve a more general problem: find the longest path from source to an arbitrary vertex (i, j) with $0 \le i \le n$, $0 \le j \le m$.

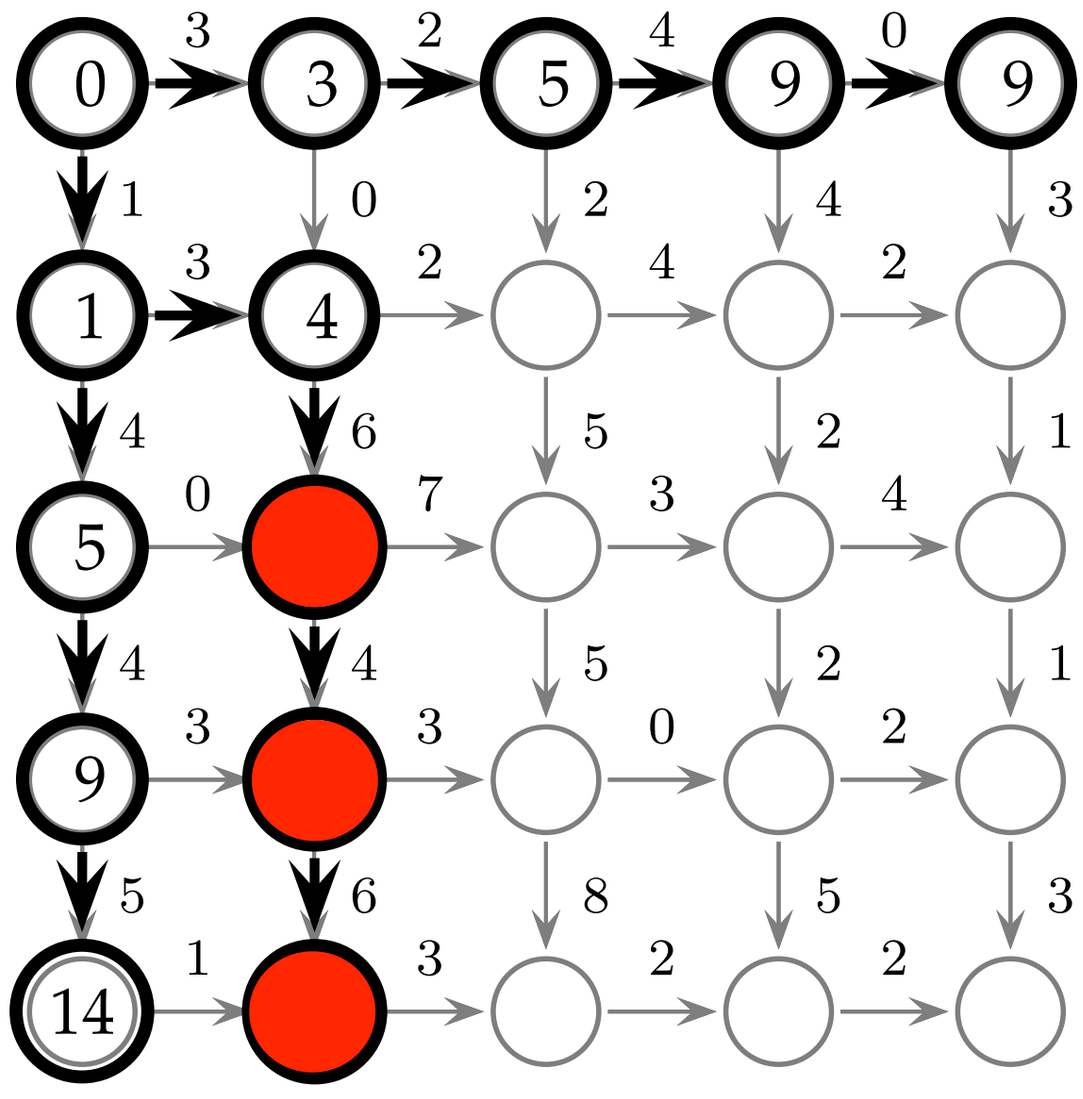
Each node's score is the maximum of the prior node scores plus the weight of the respective edge in between

The Manhattan Tourist Problem — the dynamic programming solution



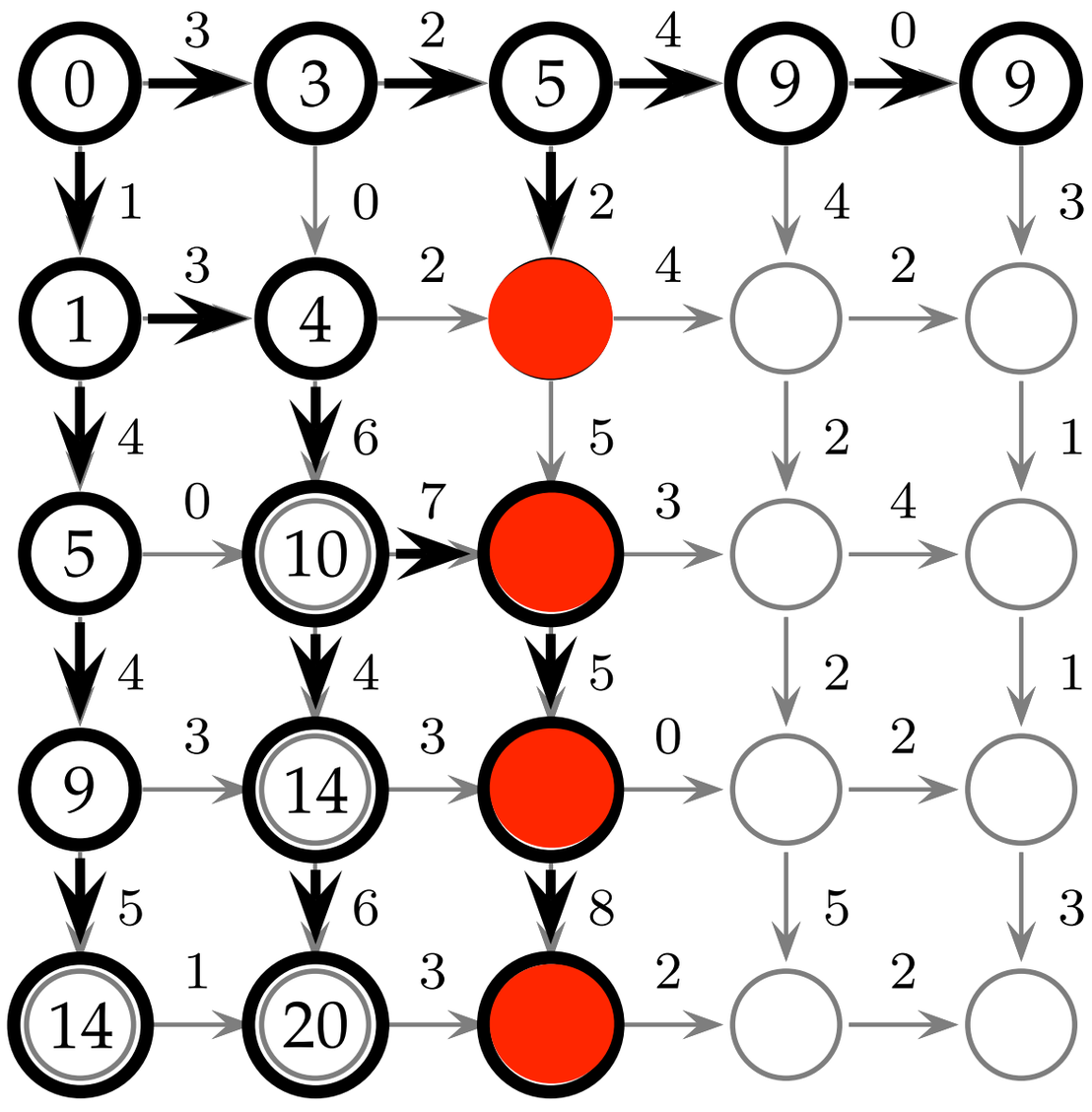
Each node's score is the maximum of the prior node scores plus the weight of the respective edge in between.

The Manhattan Tourist Problem — the dynamic programming solution



Each node’s score is the maximum of the prior node scores plus the weight of the respective edge in between.

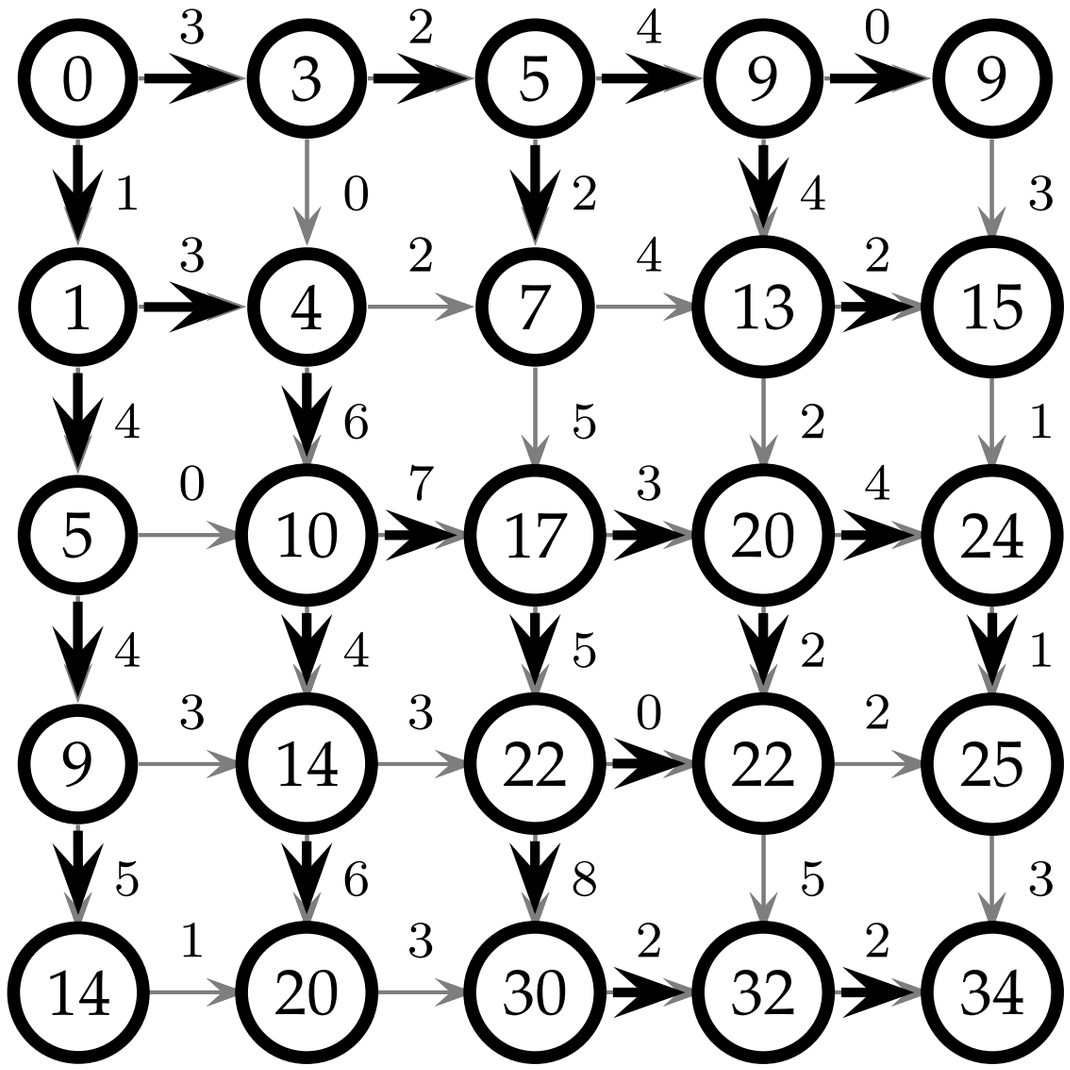
The Manhattan Tourist Problem — the dynamic programming solution



Each node's score is the maximum of the prior node scores plus the weight of the respective edge in between.

The Manhattan Tourist Problem — the dynamic programming solution

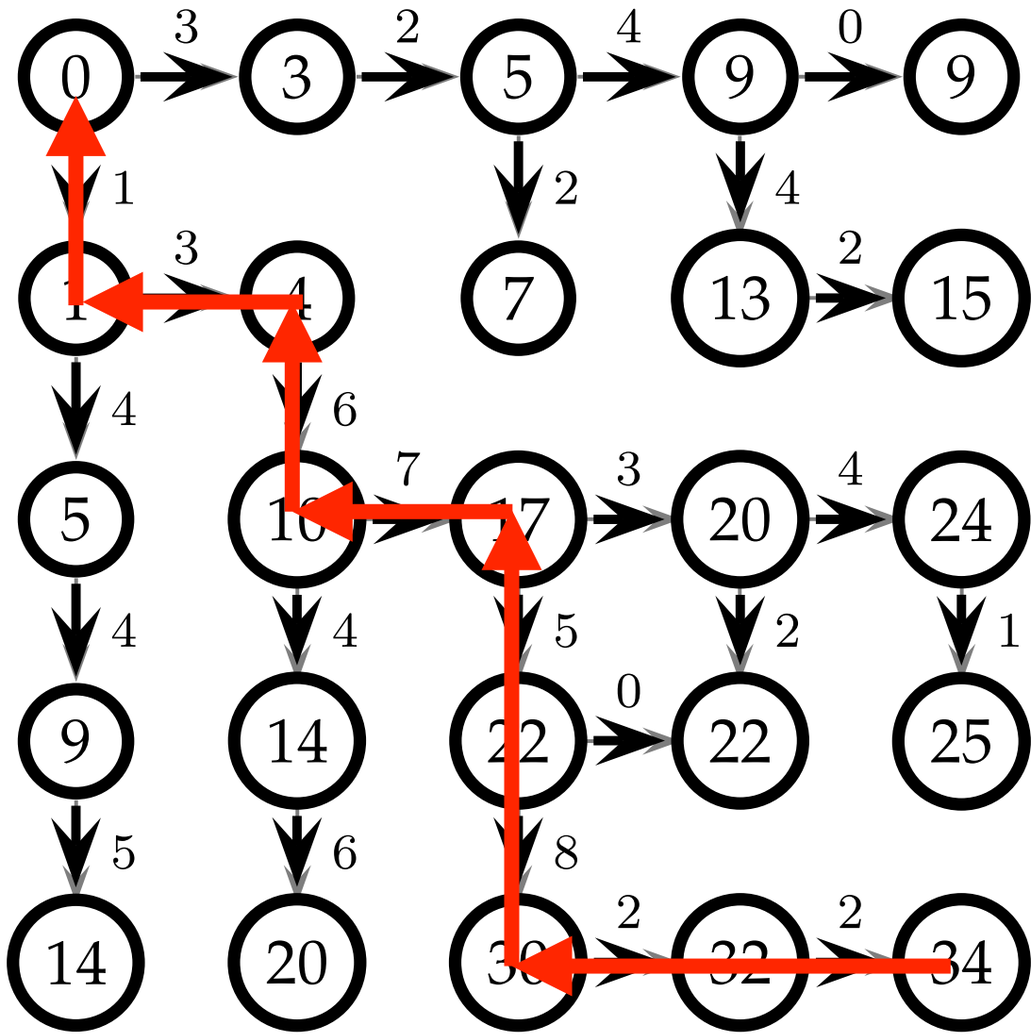
$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{ weight of the edge between } (i-1,j) \text{ and } (i,j) \\ s_{i,j-1} + \text{ weight of the edge between } (i,j-1) \text{ and } (i,j) \end{cases}$$



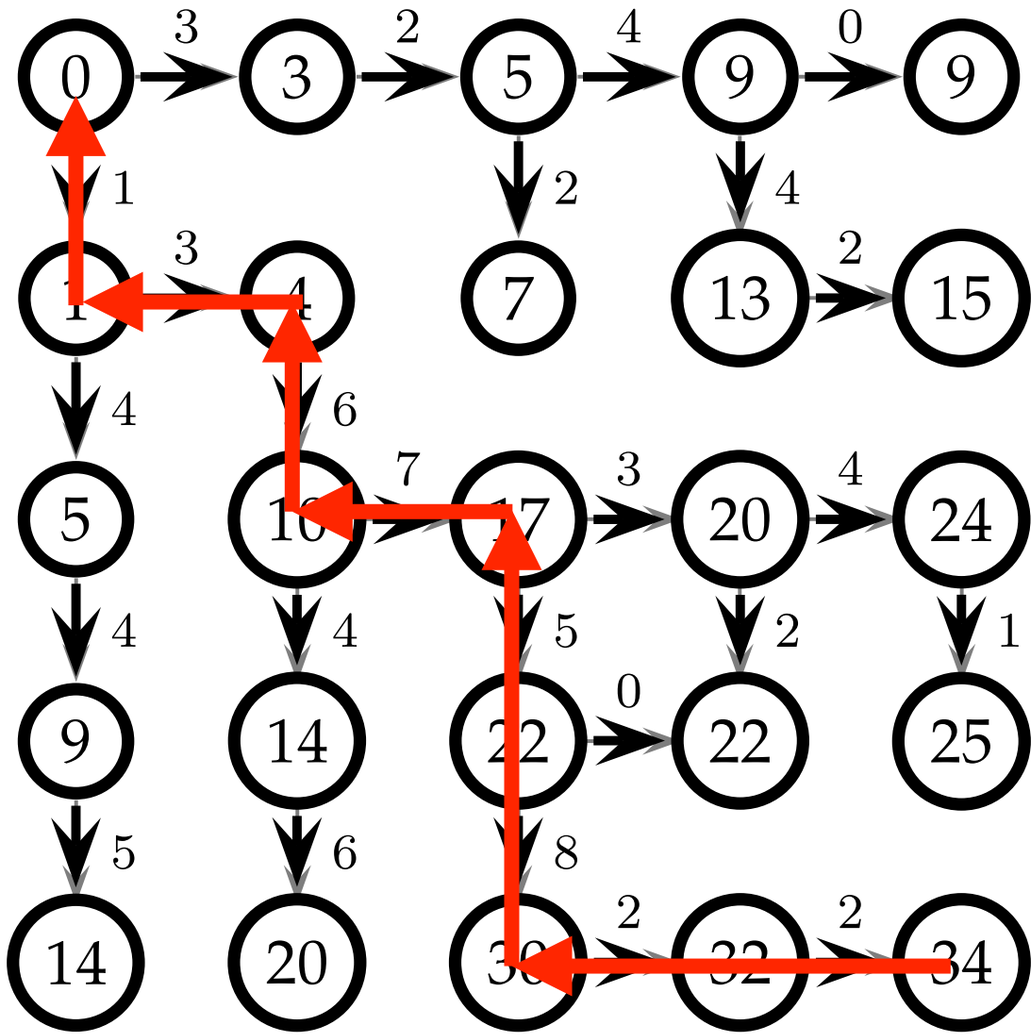
MANHATTANTOURIST($\overset{\downarrow}{\mathbf{w}}, \overset{\rightarrow}{\mathbf{w}}, n, m$)

1 $s_{0,0} \leftarrow 0$
2 **for** $i \leftarrow 1$ **to** n
3 $\quad s_{i,0} \leftarrow s_{i-1,0} + \overset{\downarrow}{w}_{i,0}$
4 **for** $j \leftarrow 1$ **to** m
5 $\quad s_{0,j} \leftarrow s_{0,j-1} + \vec{w}_{0,j}$
6 **for** $i \leftarrow 1$ **to** n
7 $\quad$ **for** $j \leftarrow 1$ **to** m
8 $\quad\quad s_{i,j} \leftarrow \max \begin{cases} s_{i-1,j} + \overset{\downarrow}{w}_{i,j} \\ s_{i,j-1} + \vec{w}_{i,j} \end{cases}$
9 **return** $s_{n,m}$

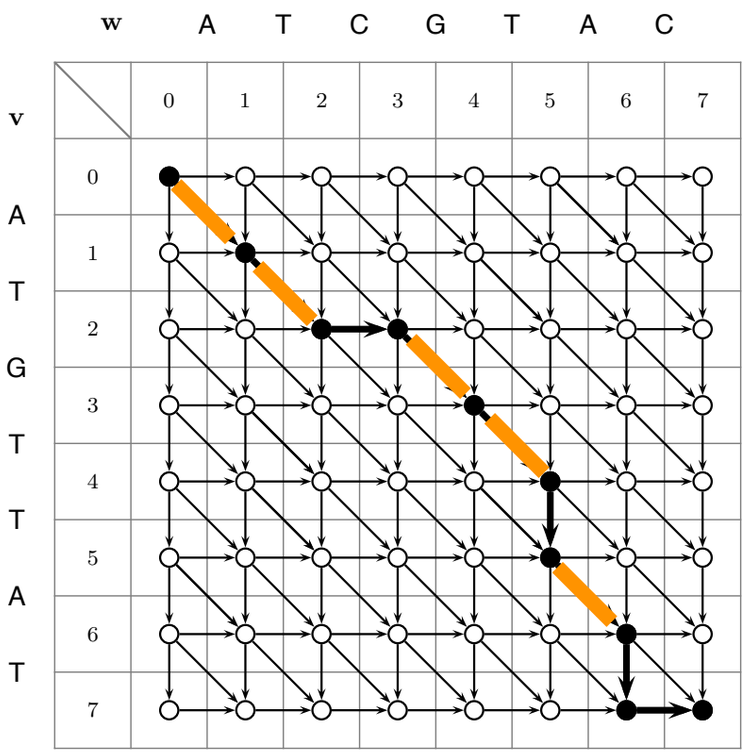
The Manhattan Tourist Problem — the dynamic programming solution

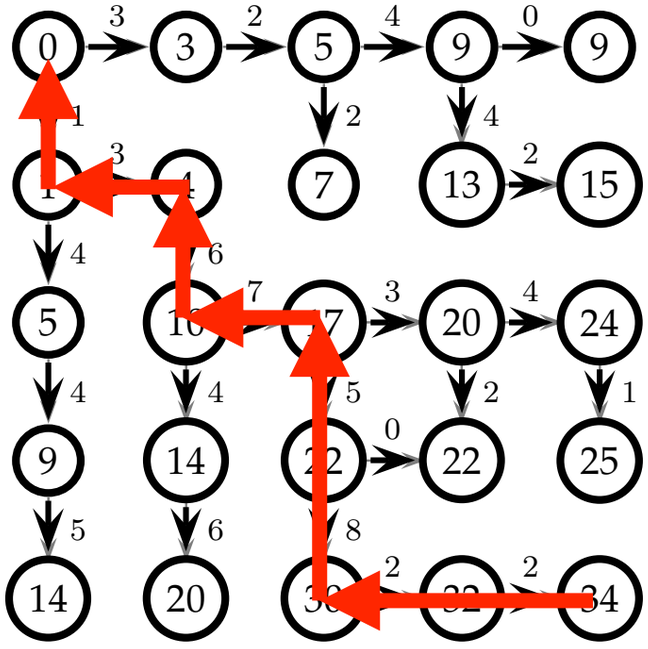


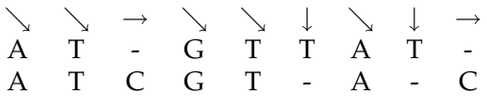
The Manhattan Tourist Problem — the dynamic programming solution



The longest common subsequence problem vs. the MTP problem





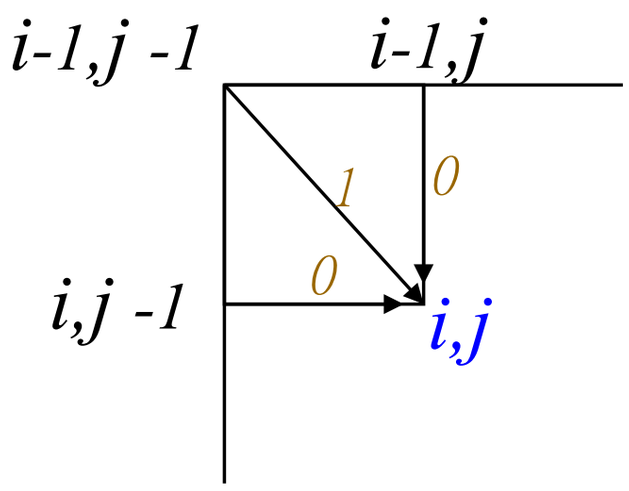


- Both can be represented as a grid. Both goal is to move through this grid in a path that the longest sequence of matching characters is found (LCS), or the maximum sum path from the top-left corner to the bottom-right corner.
- LCS can move in three directions, while MTP can move in two directions.
- The MTP problem can be solved by using dynamic programming because it has overlapping subproblems and optimal substructure.
- The LCS problem also exhibit overlapping subproblems and optimal substructure, and can be solved by dynamic programming.

The longest common subsequence problem vs. the MTP problem

MTP: $$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{ weight of the edge between } (i-1,j) \text{ and } (i,j) \\ s_{i,j-1} + \text{ weight of the edge between } (i,j-1) \text{ and } (i,j) \end{cases}$$

LCS:



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

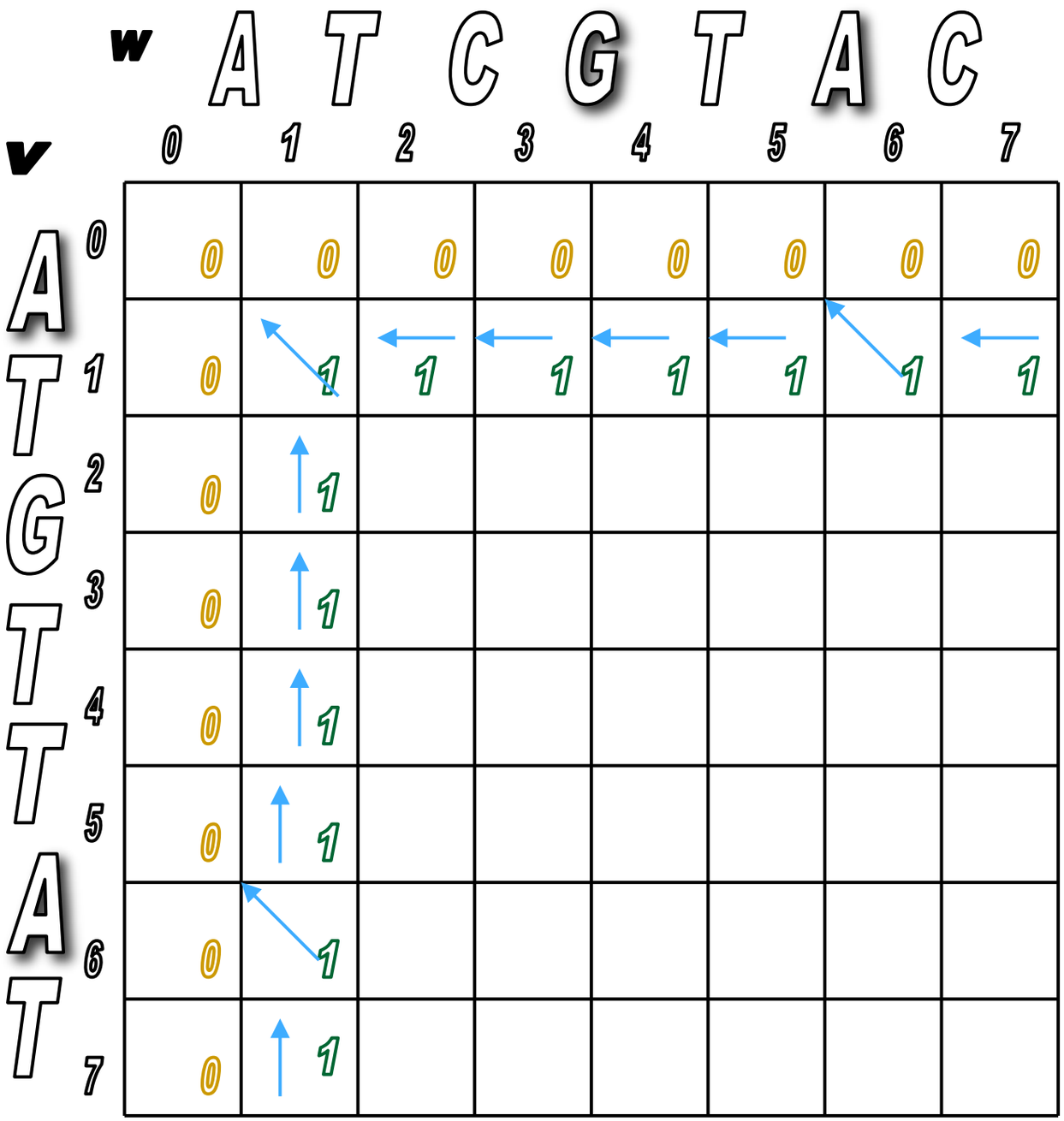
$$s_{i,j} = \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1 \text{ if } v_i = w_j \end{cases}$$

Dynamic programming approach for solving the LCS problem

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

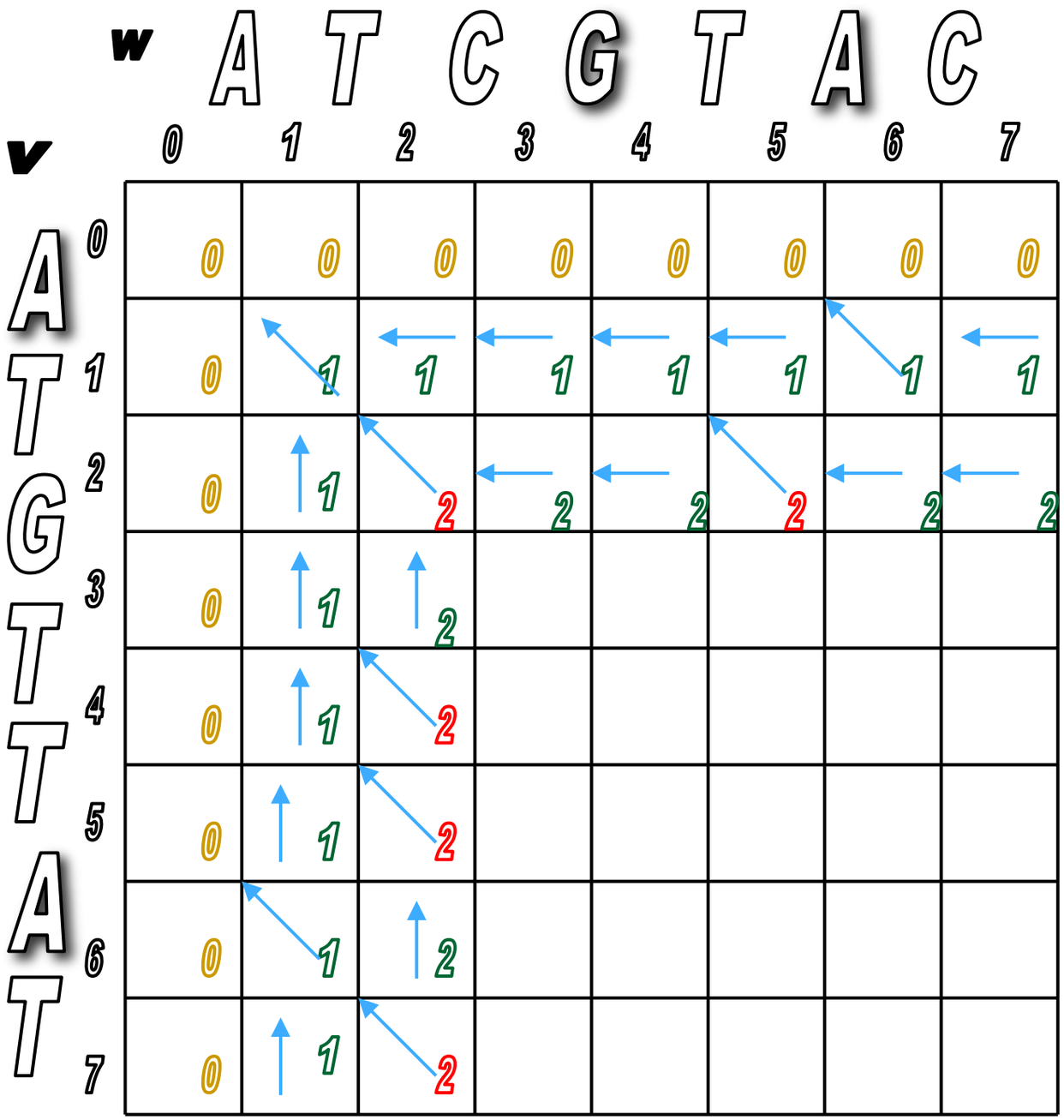
- Initialize 0th row and 0th column to be all zeroes.

Dynamic programming approach for solving the LCS problem



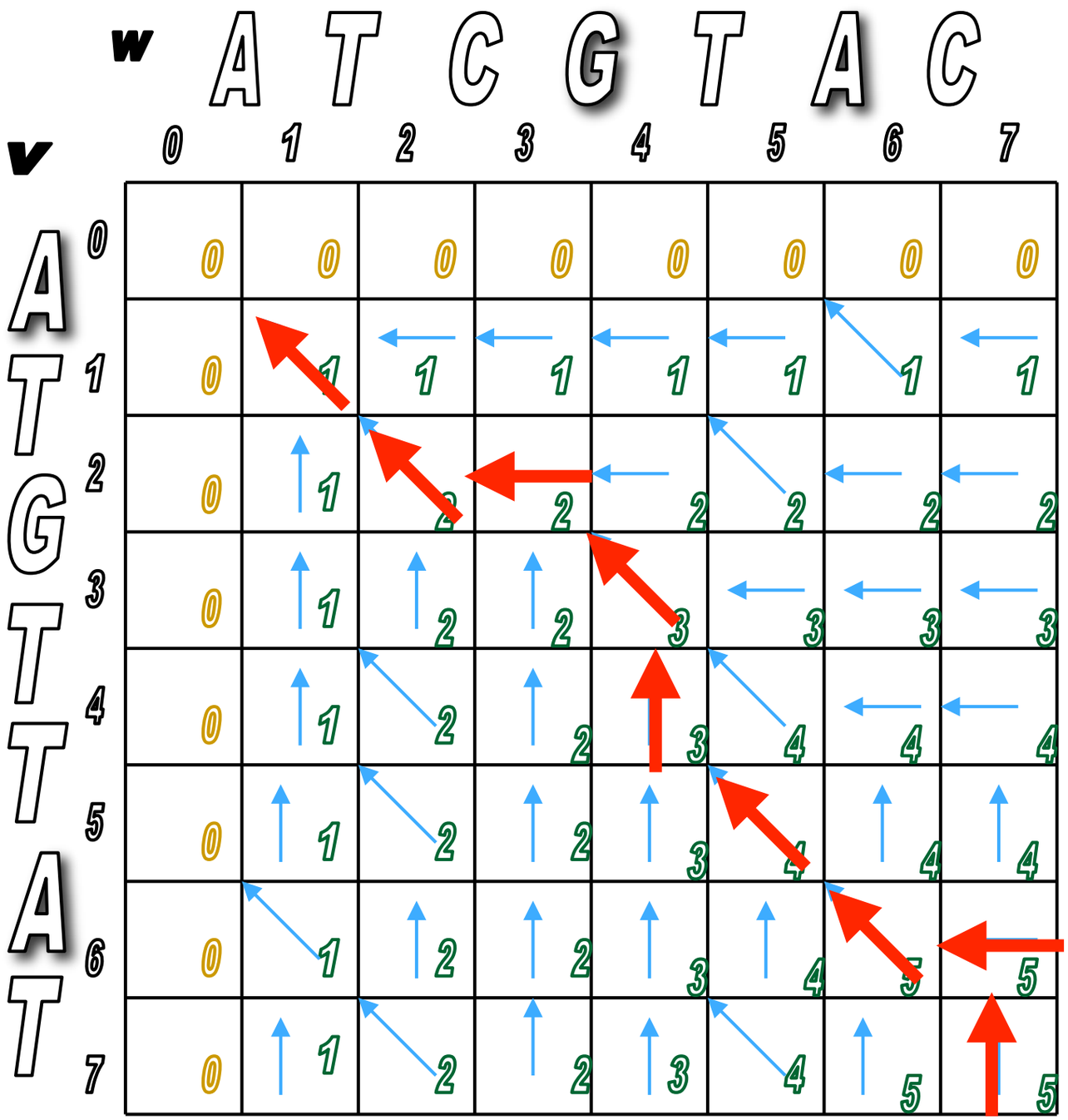
$$s_{i,j} = \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1 \text{ if } v_i = w_j \end{cases}$$

Dynamic programming approach for solving the LCS problem



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

Dynamic programming approach for solving the LCS problem



A T C G – T A C –
A T – G T T A – T

Dynamic programming approach for solving the LCS problem

LCS($\mathbf{v}, \mathbf{w}$)

1 **for** $i \leftarrow 0$ **to** n

2 $\quad s_{i,0} \leftarrow 0$

3 **for** $j \leftarrow 1$ **to** m

4 $\quad s_{0,j} \leftarrow 0$

5 **for** $i \leftarrow 1$ **to** n

6 $\quad$ **for** $j \leftarrow 1$ **to** m

7 $$s_{i,j} \leftarrow \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1, \quad \text{if } v_i = w_j \end{cases}$$

8 $$b_{i,j} \leftarrow \begin{cases} \text{“}\uparrow\text{”} & \text{if } s_{i,j} = s_{i-1,j} \\ \text{“}\leftarrow\text{”} & \text{if } s_{i,j} = s_{i,j-1} \\ \text{“}\nwarrow\text{”}, & \text{if } s_{i,j} = s_{i-1,j-1} + 1 \end{cases}$$

9 **return** $(s_{n,m}, \mathbf{b})$

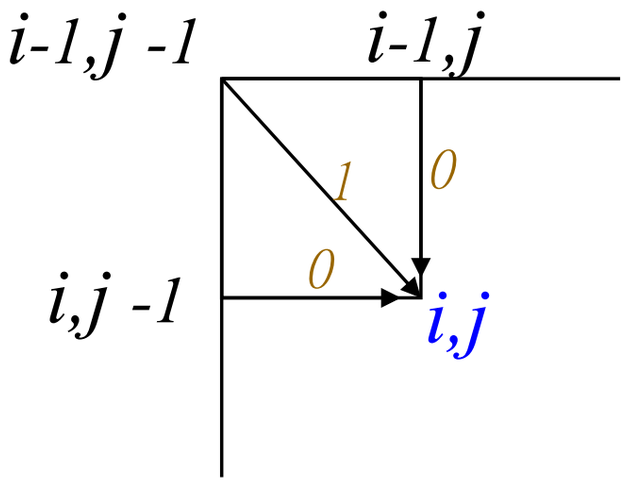
Print LCS sequences

```
PRINTLCS(b, v, i, j)
 1  if i = 0 or j = 0
 2      return
 3  if b_{i,j} = "↖"
 4      PRINTLCS(b, v, i − 1, j − 1)
 5      print v_i
 6  else
 7      if b_{i,j} = "↑"
 8          PRINTLCS(b, v, i − 1, j)
 9      else
10          PRINTLCS(b, v, i, j − 1)
```

The initial invocation that prints the solution to the problem is PRINTLCS(b, v, n, m).

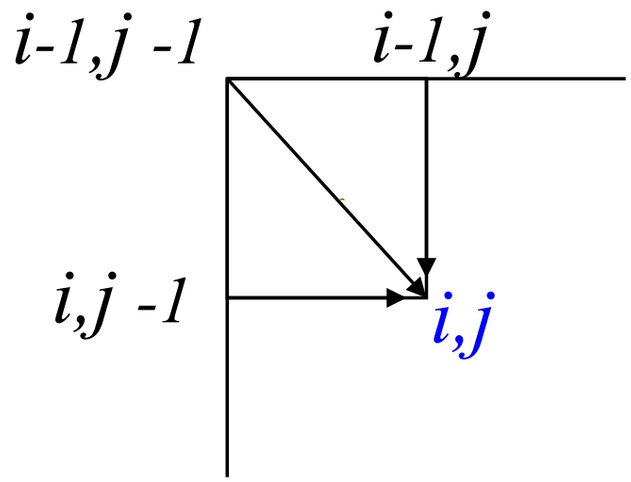
Generalize the LCS problem to the problem of global sequence alignment

LCS:



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

sequence alignment



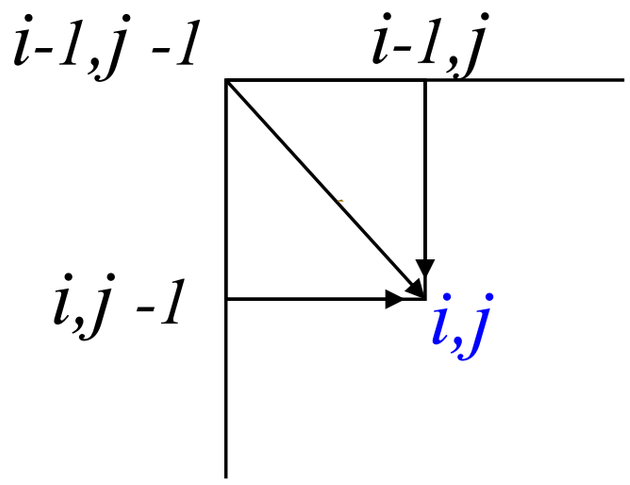
gaps (insertion/ deletion)

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$

matches/ mismatches

Generalize the LCS problem to the problem of global sequence alignment

sequence alignment



$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$

gaps (insertion/ deletion)

matches/ mismatches

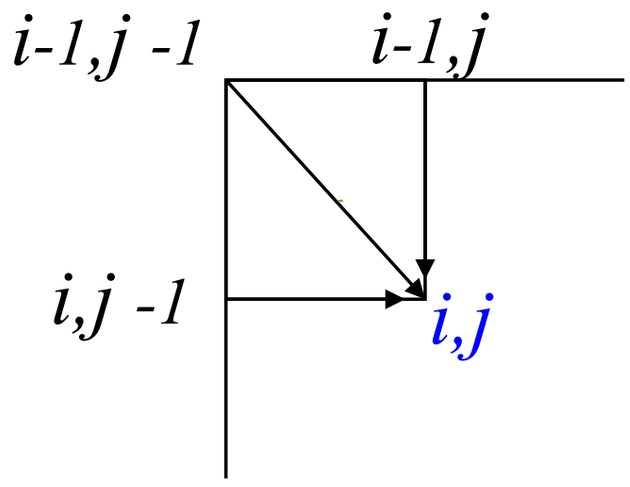
$$\#matches - \mu \cdot \#mismatches - \sigma \cdot \#indels$$

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

The LCS problem is the Global Alignment problem with the parameters μ = 0, σ = 0.

Global sequence alignment — Needleman-Wunsch algorithm

sequence alignment



gap penalties (insertion/ deletion)

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$

substitution matrix

- The Needleman-Wunsch algorithm is designed to align two sequences (such as DNA, RNA, or proteins) along their entire length, optimizing the alignment based on a scoring system that accounts for matches, mismatches, and gaps (insertions or deletions). The algorithm ensures that the alignment is "global," meaning that it considers the entire length of both sequences from start to finish.

- The concepts introduced by Needleman and Wunsch, such as scoring matrices and dynamic programming, are foundational and have been adapted in many other algorithms, including the Smith-Waterman algorithm for local alignment and more advanced techniques used in modern bioinformatics tools.

Limitations of global sequence alignment

Assumptions of Similar Lengths

- If the sequences differ significantly in length or have large, non-homologous regions, global alignment may produce misleading results by forcing alignments over the entire length, even in regions that do not correspond biologically.

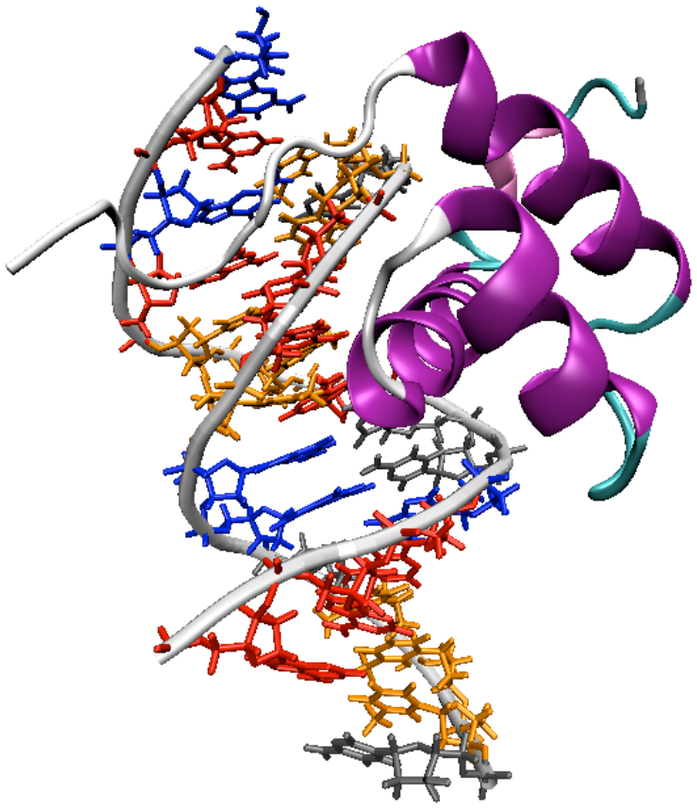
Not Suitable for Highly Divergent Sequences

- For sequences that are only partially similar or have diverged significantly (e.g., distantly related homologs), global alignment may not capture the most meaningful regions of similarity, as it tries to align the entire sequences, including poorly conserved regions.

Difficulty in Identifying Local Similarities

- It may miss or undervalue local regions of high similarity, such as conserved domains or motifs that are important but embedded within largely dissimilar sequences..

An example



- Homeobox genes have a short region called the homeodomain (60 amino acids long) that is highly conserved between species.

- A global alignment would not find the homeodomain because it would try to align the ENTIRE sequence

An example

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
  |  || |  ||  | | | |||    || |  | |  | ||||   |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG--T-CAGAT--C
```

Global alignment

```
                    tccCAGTTATGTCAGgggacacgagcatgcagagac
                       ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

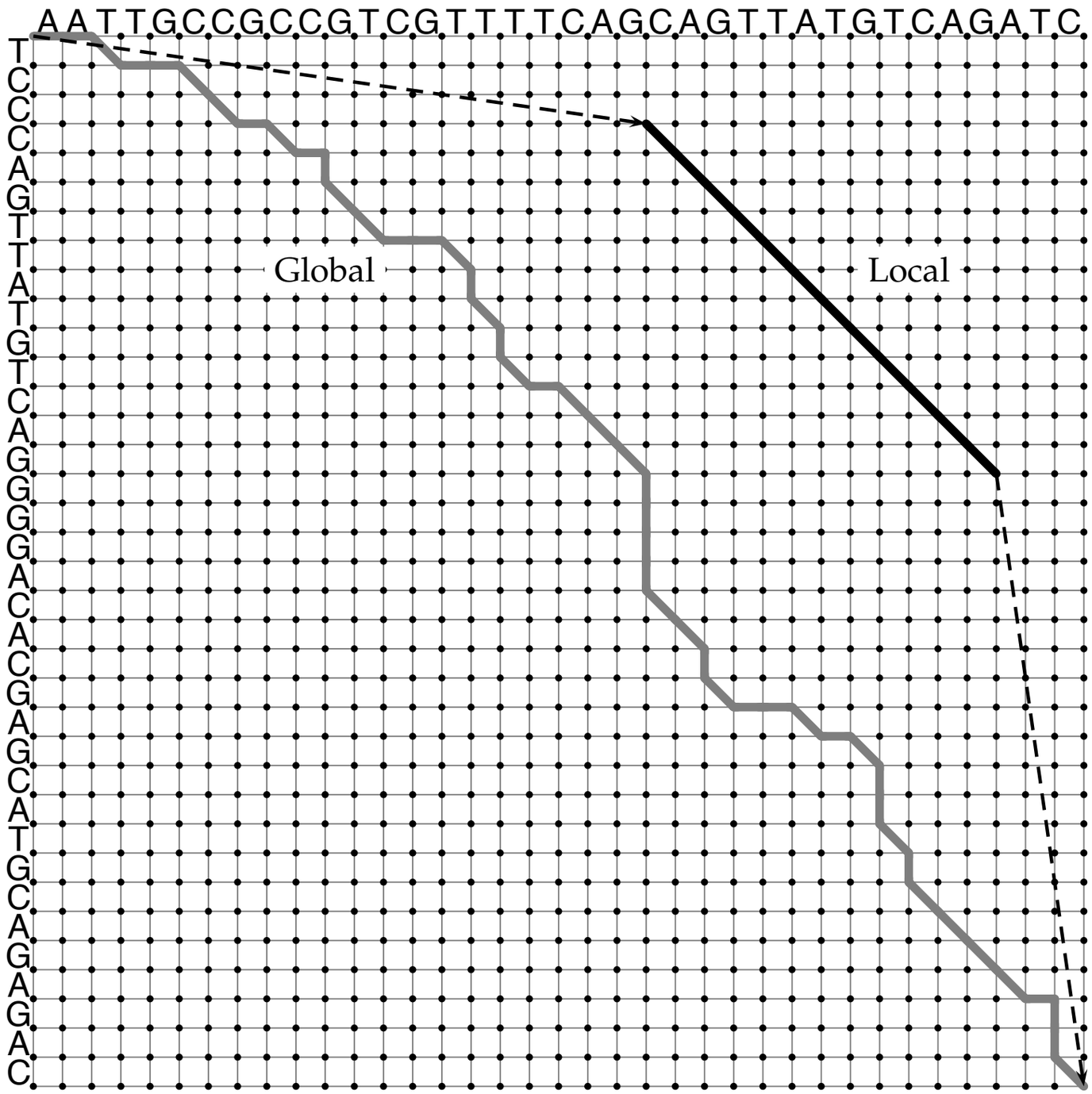
Local alignment

Local Alignment Problem:
Find the best local alignment between two strings.

Input: Strings **v** and **w** and a scoring matrix δ.

Output: Substrings of **v** and **w** whose global alignment, as defined by δ, is maximal among all global alignments of all substrings of **v** and **w**.

Global vs. local alignment



Solving the local alignment problem — the Smith-Waterman algorithm

- Smith and Waterman recognized that not all biological sequences are globally similar, but often contain regions of high similarity amidst otherwise unrelated sequence segments.

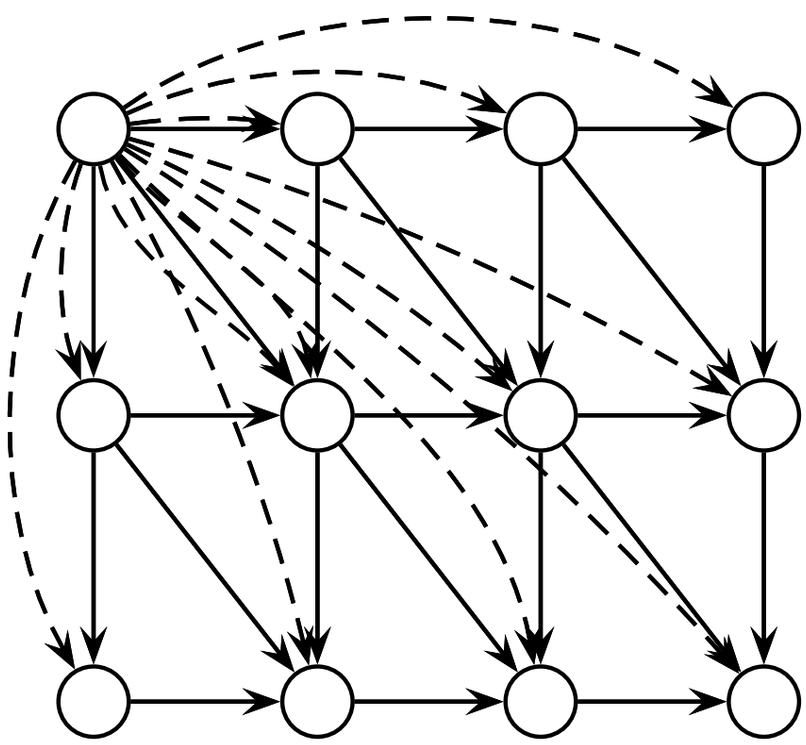
- Unlike the global alignment approach of Needleman-Wunsch, Smith-Waterman focuses on finding the best matching local subsequences between two sequences.

- Smith-Waterman's matrix can start and end at any position, offering greater flexibility in identifying biologically relevant alignments, without forcing an alignment of the entire sequences.

- The algorithm reflects a more realistic scenario in biological data, where similarities are often found within specific regions rather than across entire sequences. This is especially true for protein sequences, which may share conserved functional domains while differing in other regions.

Solving the local alignment problem — the Smith-Waterman algorithm

- Local alignment: the Smith-Waterman algorithm (aligning segments of two sequences with the highest density of matches)



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

Figure 6.17 The Smith-Waterman local alignment algorithm introduces edges of weight 0 (here shown with dashed lines) from the source vertex $(0, 0)$ to every other vertex in the edit graph.

Key differences in dynamic programming matrices for local and global sequence alignment

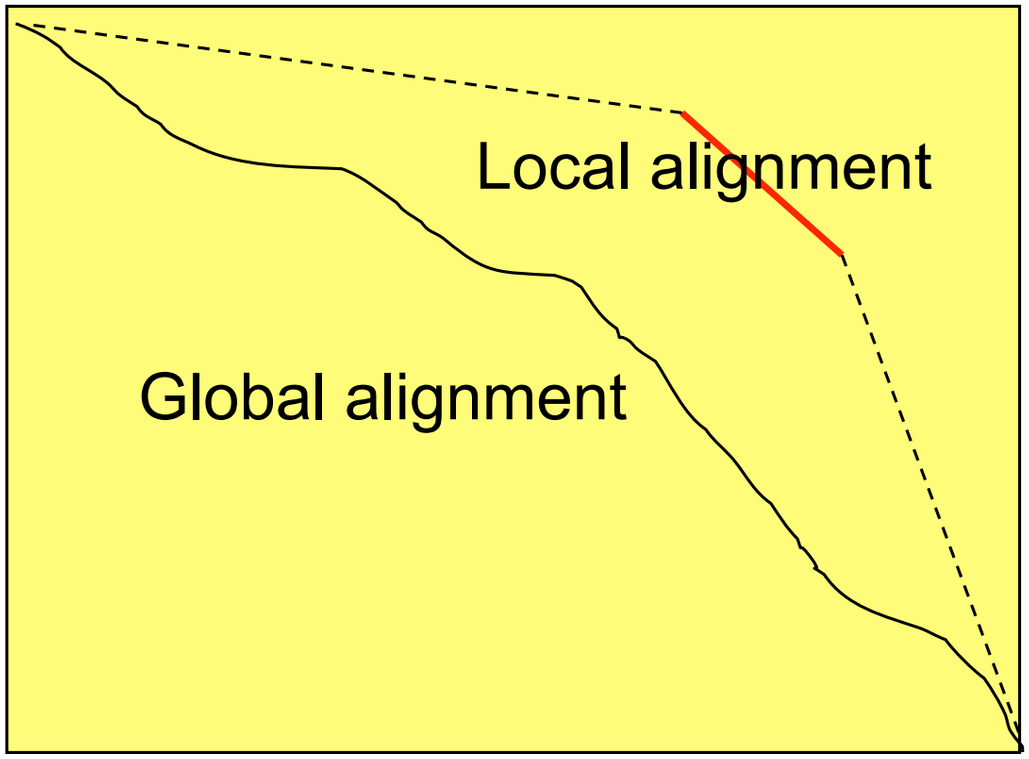
- Both the SW (local) and the NW (global) algorithms use a dynamic programming matrix, but with key differences in how the matrix is initialized and filled:

  - **Initialization:** The SW matrix is initialized with zeros, unlike NW alignment, where the matrix is initialized based on gap penalties.

  - **Matrix Filling:** For each cell in the matrix, the SW algorithm considers three potential scores (from the left, above, or diagonal cells) plus a fourth option of starting a new alignment (hence adding zero). This allows the alignment to start and end at any position, enabling the detection of local alignments.

- The SW's traceback process starts from the highest-scoring cell in the matrix and moves back to the first zero encountered. This identifies the subsequence that produces the best local alignment.

local vs. global sequence alignment



The Global Alignment tries to find the longest path between vertices (0,0) and (n,m) in the 2D graph.

The Local Alignment tries to find the longest path among paths between arbitrary vertices (i,j) and (i’, j’) in the 2D graph.

The largest value of Si,j over the whole 2D graph is the score of the best local alignment.

Example of a local sequence alignment

Scoring Metric:

Match: $s(a_i, b_j) = 1$

Mismatch: $s(a_i, b_j) = -1$

Gap: -2 penalty

Maximum of possible scores:

(a) 0 + s(A,A) = 0 + 1 = 1

(b) 0 - g = 0 - 2 = -2

(c) 0 - g = 0 - 2 = -2

(d) 0 (no pointer)

Option (a) gives the maximum score so this value is placed in the matrix, and **option (a) pointer** is retained

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

a

Example of a local sequence alignment

Scoring Metric:
Match: $s(a_i, b_j) = 1$
Mismatch: $s(a_i, b_j) = -1$
Gap: -2 penalty

Maximum of possible scores:
(a) 0 + s(A,A) = 0 + 1 = 1
(b) 1 - g = 1 - 2 = -1
(c) 0 - g = 0 - 2 = -2
(d) 0 (no pointer)

Option (a) gives the maximum score so this value is placed in the matrix, and **option (a) pointer** is retained

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

a

Example of a local sequence alignment

Scoring Metric:

Match: $s(a_i, b_j) = 1$

Mismatch: $s(a_i, b_j) = -1$

Gap: -2 penalty

Maximum of possible scores:

(a) 0 + s(A,T) = 0 - 1 = -1

(b) 1 - g = 1 - 2 = -1

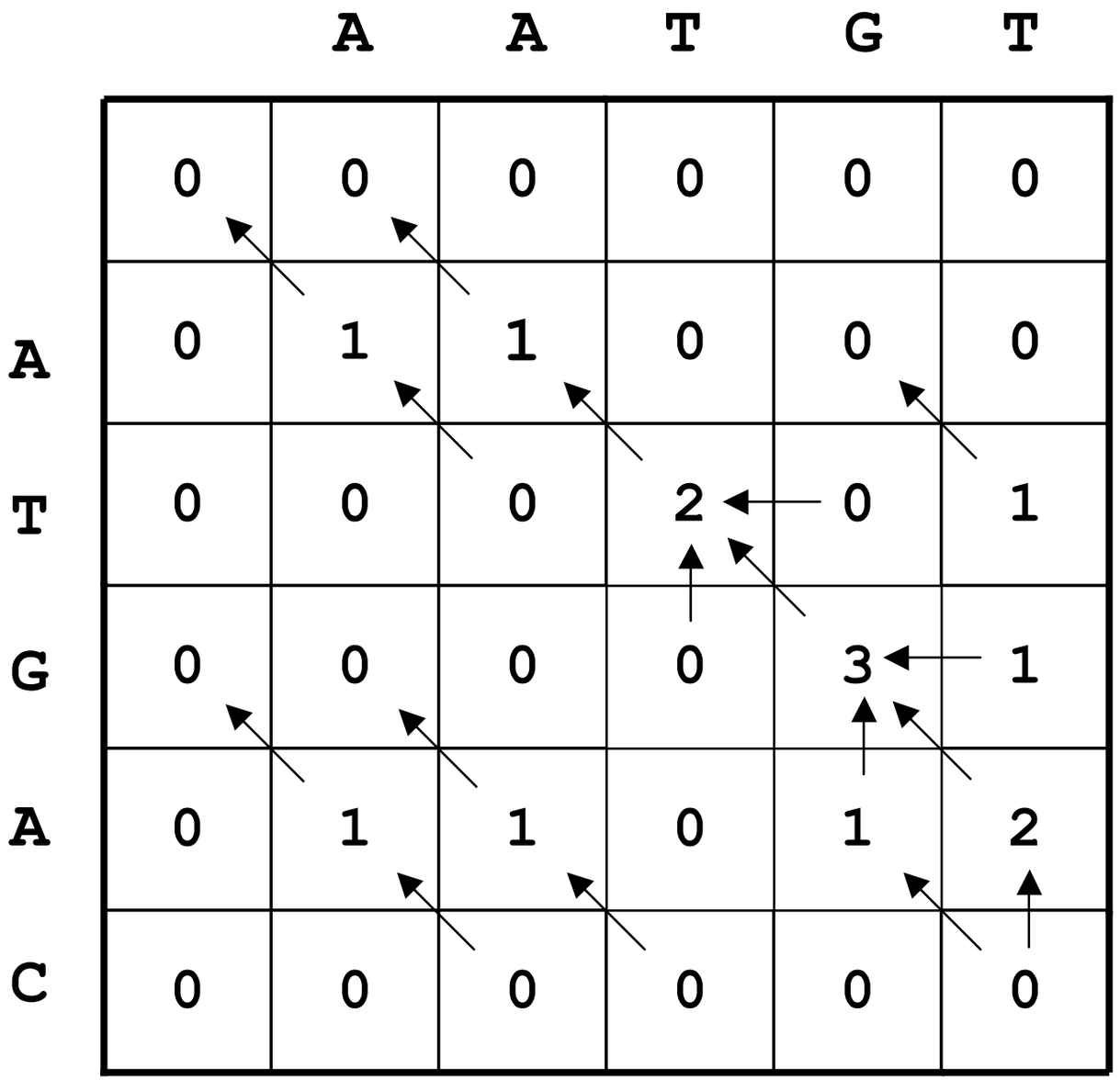
(c) 0 - g = 0 - 2 = -2

(d) 0 (no pointer)

Option (d) gives the maximum score so this value is placed in the matrix, and **no pointer is retained**

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

Example of a local sequence alignment



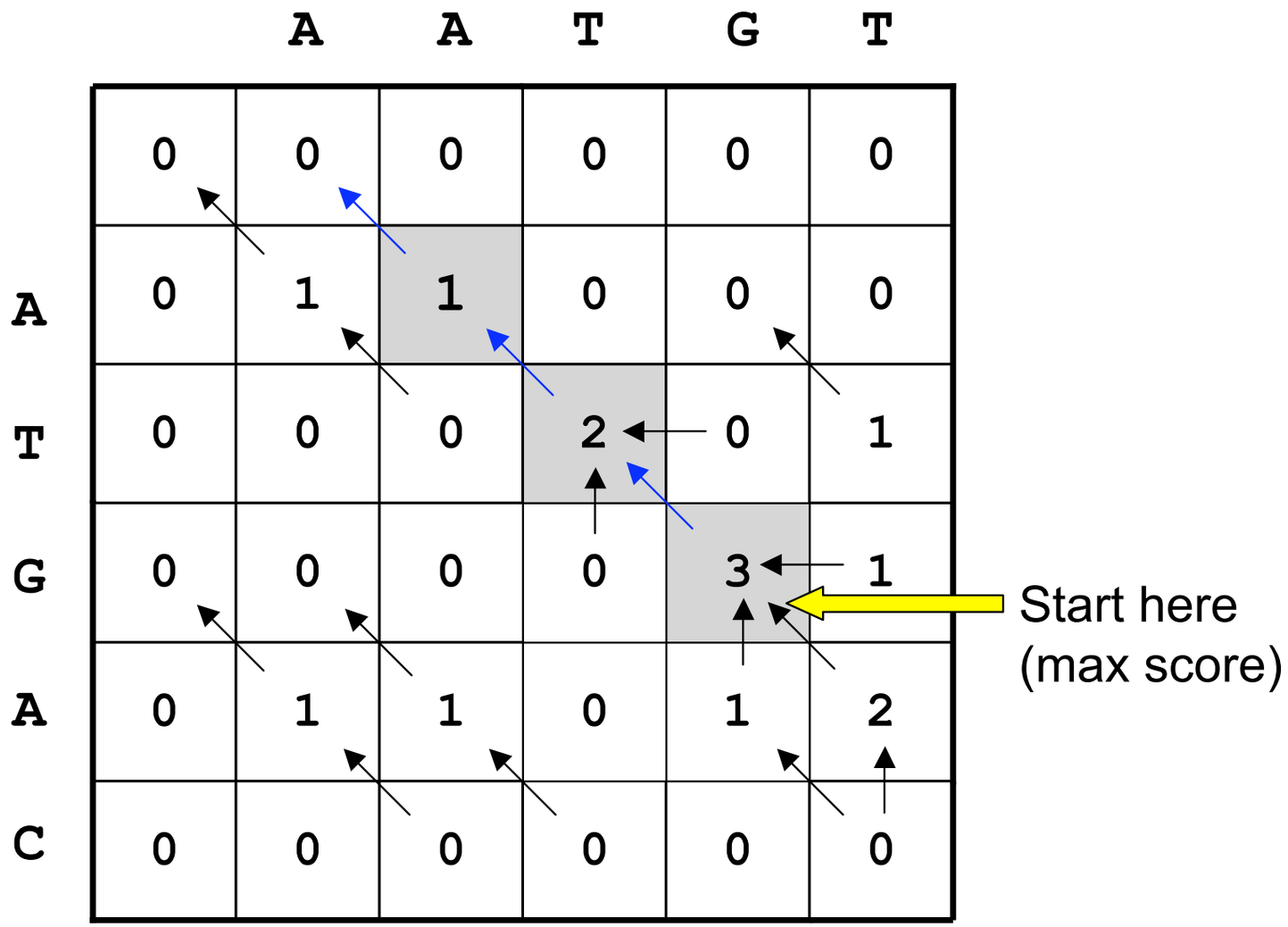
Example of a local sequence alignment

§ Begin with maximum scoring element

§ Follow pointers that gave max. score for each element

§ Continue until reach an element with zero score

§ Construct alignment from traceback path



Local Alignment shown in blue:

AATGT
ATGAC

The complexity of pairwise sequence alignment

LCS

Global

Local

LCS$(\mathbf{v}, \mathbf{w})$
1 **for** $i \leftarrow 0$ **to** n
2 $\quad s_{i,0} \leftarrow 0$
3 **for** $j \leftarrow 1$ **to** m
4 $\quad s_{0,j} \leftarrow 0$
5 **for** $i \leftarrow 1$ **to** n
6 $\quad$ **for** $j \leftarrow 1$ **to** m
7 $\quad\quad s_{i,j} \leftarrow \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1, \quad \text{if } v_i = w_j \end{cases}$
8 $\quad\quad b_{i,j} \leftarrow \begin{cases} \text{“}\uparrow\text{”} & \text{if } s_{i,j} = s_{i-1,j} \\ \text{“}\leftarrow\text{”} & \text{if } s_{i,j} = s_{i,j-1} \\ \text{“}\nwarrow\text{”}, & \text{if } s_{i,j} = s_{i-1,j-1} + 1 \end{cases}$
9 **return** $(s_{n,m}, \mathbf{b})$

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$

The time complexity: O(M × N).
The space complexity: O(M × N).

Sequence identity and sequence similarity

$$\text{Sequence Identity (\%)} = \left(\frac{\text{Number of Identical Matches}}{\text{Total Length of Alignment}} \right) \times 100$$

Sequence 1: AGCTGA
Sequence 2: AGCTAA

$$\text{Sequence Identity (\%)} = \left(\frac{5}{6} \right) \times 100 = 83.33\%$$

$$\text{Sequence Similarity (\%)} = \left(\frac{\text{Number of Identical Matches} + \text{Conservative Substitutions}}{\text{Total Length of Alignment}} \right) \times 100$$

Sequence 1: A G C T G A
Sequence 2: A G S T G V

$$\text{Sequence Similarity (\%)} = \left(\frac{5}{6} \right) \times 100 = 83.3\%$$

Sequence identity

S1: AGCTGA
S2: A- GTCA

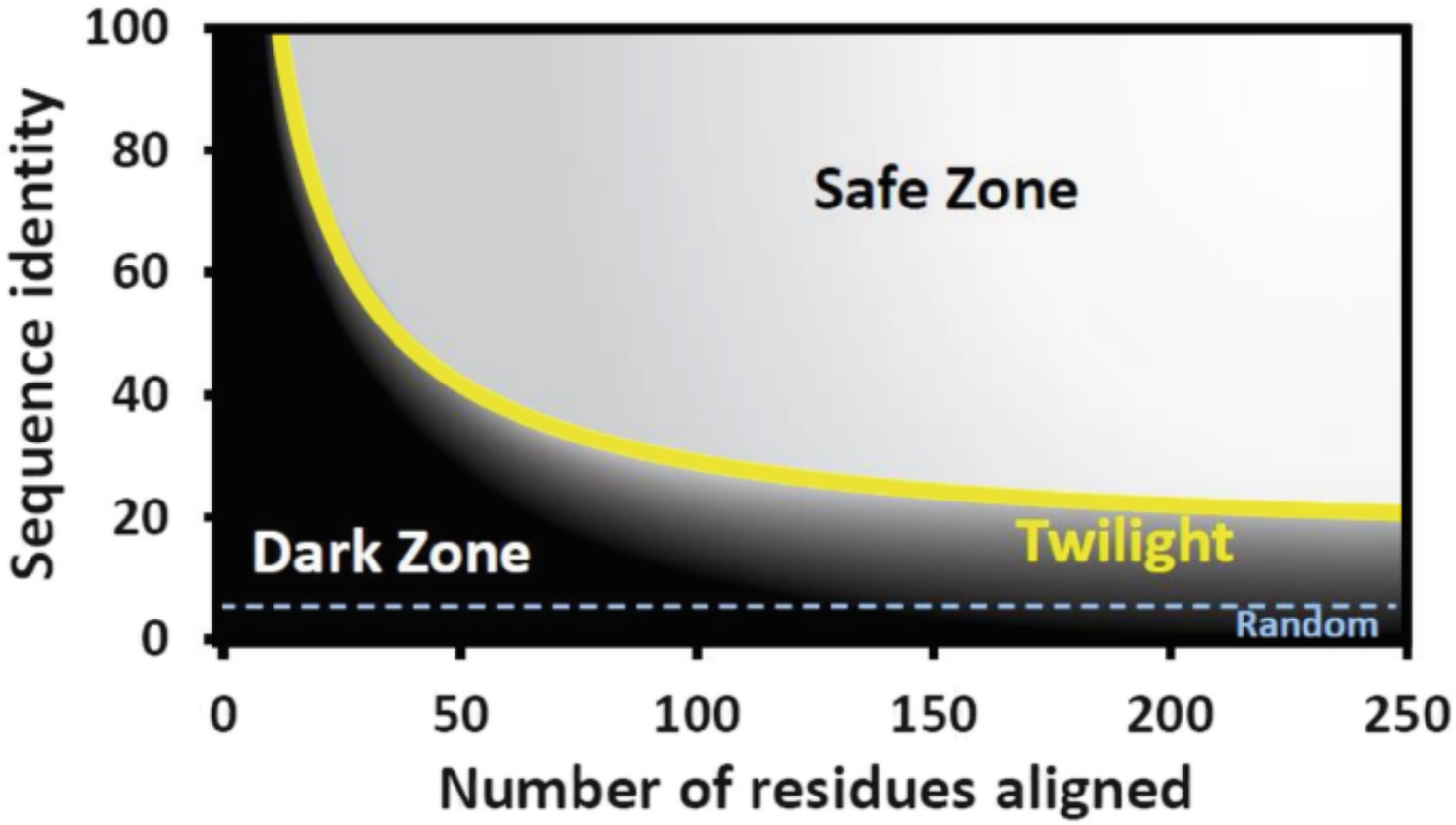
$$\text{Sequence Identity (\%)} = \left(\frac{\text{Number of Identical Matches}}{\text{Total Length of Alignment (Including Gaps)}} \right) \times 100$$

$$\boxed{\text{Sequence Identity (\%)} = \left(\frac{4}{6} \right) \times 100 = 66.67\%}$$

$$\text{Sequence Identity (\%)} = \left(\frac{\text{Number of Identical Matches}}{\text{Number of Aligned Positions (Excluding Gaps)}} \right) \times 100$$

$$\boxed{\text{Sequence Identity (\%)} = \left(\frac{4}{5} \right) \times 100 = 80\%}$$

Sequence identity and the twilight-zone



- **Sequence Identity** is a key factor in inferring homology, with higher identity suggesting a stronger evolutionary relationship.

- The **twilight zone** (20-30% identity) represents a range where homology inference is uncertain, requiring supplementary evidence to confirm evolutionary relationships.

- The twilight zone remains a useful concept for understanding the limits of traditional sequence comparison methods, but it is now complemented by more powerful tools that can detect homology even when sequence identity is low.

Challenges in pairwise sequence alignment

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$

Appropriate gap penalties

Appropriate substitution matrices

Complexity in computation