Sequence Comparison

- Mutation in DNA is a natural evolutionary process.

- Similarity between DNA sequences can be a clue to a common evolutionary orgin.

- Levenshtein Edit Distance between strings: Mininum number of edit operations to transform one string into another.

- Edit Operations:

  - Insert a symbol.

  - Delete a symbol.

- Dynamic Programming algorithm to compute edit distance between two strings.

Definition 1 (Distance) *A distance on a set E is a function $d : E \times E \to R$ such that:*

1. $d(x, x) = 0 \; \forall x \in E$ *and* $d(x, y) > 0$ *for* $x \neq y$.

2. $d(x, y) = d(y, x) \; \forall x, y \in E$ *(d is symmetric).*

3. $d(x, y) \leq d(x, z) + d(y, z) \; \forall x, y, z \in E$ *(triangle inequality).*

- If edit distance limited to insertions and deletions, equivalent to Longest Common Subsequence (LCS) problem.

- Biological term is an *alignment*.

- Alignment of strings V and W is a two-row matrix such that first (second) row contains charcters of V (W) in order, interspersed with some spaces.

- Score of an alignment is sum of scores of its columns.

- Usually, positive for matching letters, and negative for distinct letters.

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

- Alignments can be local or global.
- Global: Similarity between whole strings.
- Local: Similarity between substrings.
- GenBank: 10^9 sequences and counting.
- Compare sequence of length 10^3.
- Filtering: Search for short substrings and use them as seeds for subsequent searches.
- FASTA and BLAST.

Longest Common Subsequence Problem

Definition 2 (Common Subsequence) *Given strings* $V = v_1 \dots v_n$ *and* $W = w_1 \dots w_m$ *as sequences of indices*

$$1 \leq i_1 < \dots < i_k \leq n,$$

and

$$1 \leq j_1 < \dots < j_k \leq m,$$

such that

$$v_{i_t} = w_{j_t} \text{ for } 1 \leq t \leq k$$

.

Let $s(V, W)$ be the length k of the LCS, then, $d(V, W) = n + m - 2s(V, W)$ is the minimum number of insertions and deletions to transform $V \rightarrow W$.

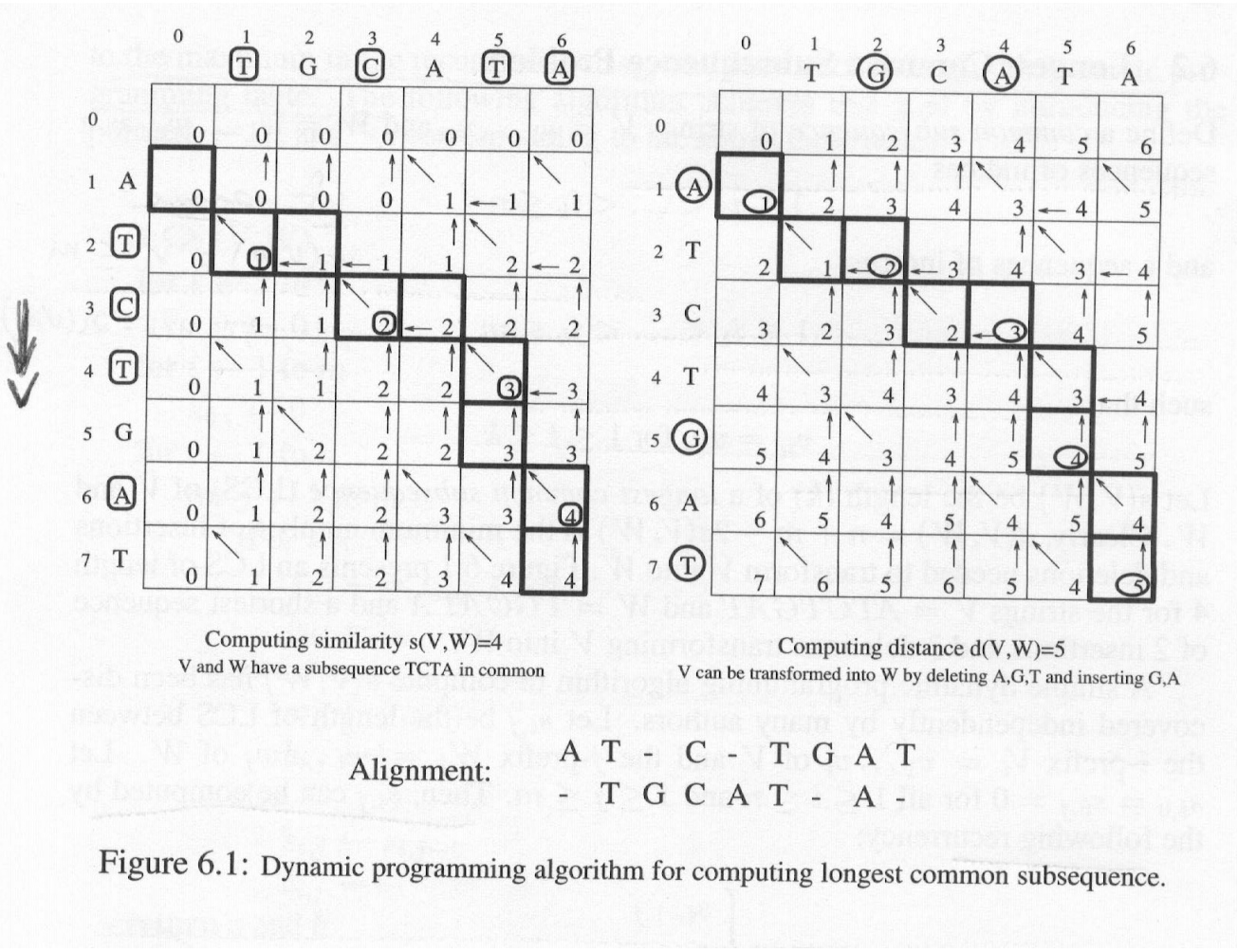

Figure 6.1: Dynamic programming algorithm for computing longest common subsequence.

Dynamic Programming

- Algorithm to compute $s(V, W)$.
- $s_{i,j}$ is length of LCS between i-prefix $V_i = v_1 \dots v_i$ of V and j-prefix $W_j = w_1 \dots w_j$ of W.
- $s_{i,0} = s_{0,j} = 0 \forall 1 \leq i \leq n$ and $1 \leq j \leq m$.
- Compute $s_{i,j}$ recursively

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

- First (second) term is when v_i (w_j) is not present in LCS of V_i and W_j.

- Third term is when both v_i and w_j are present in LCS of V_i and W_j.

- Or when v_i matches w_j.

Edit Distance

Edit distance can also be computed by dynamic programming with initial conditions $d_{i,0} = i, d_{0,j} = j \ \forall \ 1 \leq i \leq n$ and $1 \leq j \leq m$.

$$d_{i,j} = \min \begin{cases} d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \\ d_{i-1,j-1} \quad \text{if } v_i = w_j \end{cases}$$

Backtracking

- Remember where you came from with pointer.
- Backtrack through matrix to find LCS.
- to get V substitute v_i for i-th occurence of $\uparrow$ or $\nwarrow$, and by replacing all occurrences of $\leftarrow$ with $-$.
- to get W substitute w_j for j-th occurence of $\leftarrow$ or $nwarrow$, and by replacing all occurrences of $\uparrow$ with $-$.

```
LCS (V, W)
for i ← 1 to n
    s_{i,0} ← 0
for i ← 1 to m
    s_{0,i} ← 0
for i ← 1 to n
    for j ← 1 to m
        if v_i = w_j
            s_{i,j} ← s_{i-1,j-1} + 1
            b_{i,j} ←↖
        else if s_{i-1,j} ≥ s_{i,j-1}
            s_{i,j} ← s_{i-1,j}
            b_{i,j} ←↑
        else
            s_{i,j} ← s_{i,j-1}
            b_{i,j} ←←
return s and b
```

```
PRINT-LCS (b, V, i, j)
if i = 0 or j = 0
    return
if b_{i,j} = ↖
    PRINT-LCS(b, V, i − 1, j − 1)
    print v_i
else if b_{i,j} = ↑
    PRINT-LCS(b, V, i − 1, j)
else
    PRINT-LCS(b, V, i, j − 1)
```

Sequence Alignment

- Extend a k-letter alphabet with a space '-': $\mathcal{A}' = \mathcal{A} \cup \{-\}$

- An alignment of V and W is a $2 \times l$ matrix $A(l \geq n, m)$ with first and second rows containing V and W, respectively, interspersed with spaces.

- No column contains two spaces.

- Columns with space are called *indels*.

- Columns with space in first (second) row are called insertions (deletions).

- Columns with same letter are matches.

- Columns with different letters are mismatches.

- Score of a column is similarity score $\delta(x, y)$, for every pair of symbols $x, y \in \mathcal{A}$.

- Score of alignment is sum of score of its columns.

$$\begin{aligned}\delta(x, x) &= 1 \\ \delta(x, -) &= -\sigma \\ \delta(-, x) &= -\sigma \\ \delta(x, y) &= -\mu\end{aligned}$$

Global Alignment

Global sequence alignment is to find the alignment of sequences with a maximal score.

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

Every alignment corresponds to a path in the edit graph, where sequence alignment is equivalent to finding the longest path from source to sink in directed acyclic graph.

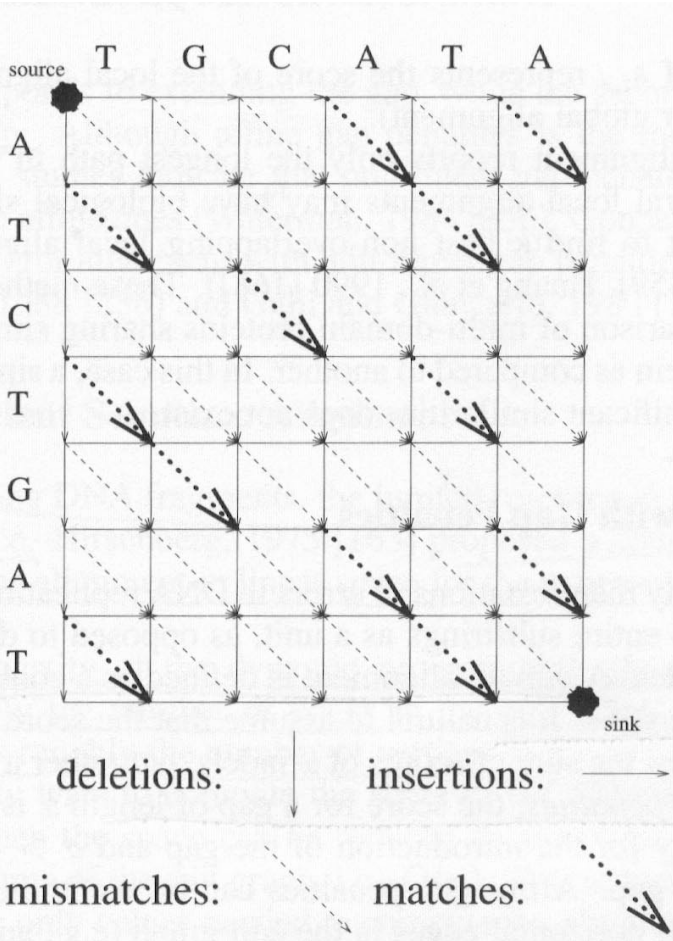

Figure 6.2: Edit graph. The weights of insertion and deletion edges are $-\sigma$, the weights of mismatch edges are $-\mu$, and the weights of match edges are 1.

Local Alignment

Biologically relevant sequences in part of DNA fragment, but not others. Then, goal is to maximize similarity over substrings. Find maximum entry in whole array which will correspond to optimal local alignment.

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

Add edges of weight 0 from source to every other vertex. Provide a free jump to arbitrary starting place.

Example

$$D_{i,j} = \min \begin{cases} D_{i-1,j} + w(v_i, -) \\ D_{i,j-1} + w(-, w_j) \\ D_{i-1,j-1} + w(v_i, w_j) \end{cases}$$

Consider the words $V = AT$ and $W = AAGT$, and a cost function where $\forall\, x \neq y :\ w(x, y) = 1$ and $w(x, x) = 0$. Then the distance matrix is:

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

The trace matrix is:

		A	A	G	T
	0	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
A	$\uparrow$	$\nwarrow$	$\leftarrow, \nwarrow$	$\leftarrow$	$\leftarrow$
T	$\uparrow$	$\uparrow$	$\leftarrow$	$\leftarrow, \nwarrow$	$\nwarrow$

Two possible alignments are:

A	-	-	T
A	A	G	T

-	A	-	T
A	A	G	T