

Sequence Alignment

String Similarity

How similar are two strings?

- **ocurrence**
- **occurrence**

o	c	u	r	r	a	n	c	e	-
o	c	c	u	r	r	e	n	c	e

6 mismatches, 1 gap

o	c	-	u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e

1 mismatch, 1 gap

o	c	-	u	r	r	-	a	n	c	e
o	c	c	u	r	r	e	-	n	c	e

0 mismatches, 3 gaps

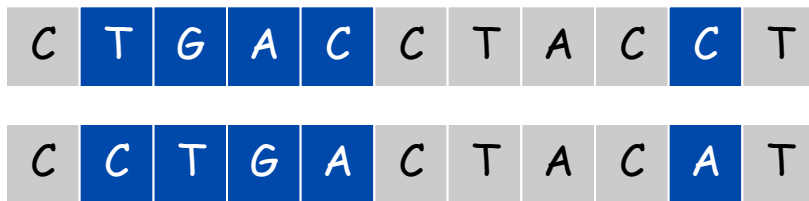
Edit Distance

Applications.

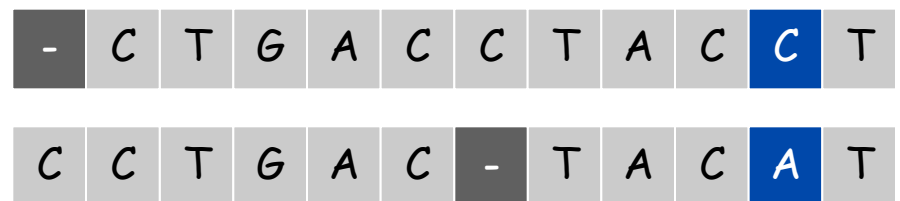
- Basis for Unix diff.
- Speech recognition.
- Spell-checking.
- Computational biology.

Edit distance. [Levenshtein 1966, Needleman-Wunsch 1970]

- Gap penalty δ ; mismatch penalty α_{pq} .
- Cost = sum of gap and mismatch penalties.



$$\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA}$$



$$2\delta + \alpha_{CA}$$

Sequence Alignment

Goal: Given two strings $X = x_1 x_2 \dots x_m$ and $Y = y_1 y_2 \dots y_n$ find alignment of minimum cost.

Def. An **alignment** M is a set of ordered pairs $x_i - y_j$ such that each item occurs in at most one pair and no crossings.

Def. The pair $x_i - y_j$ and $x_{i'} - y_{j'}$ **cross** if $i < i'$, but $j > j'$.

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i: x_i \text{ unmatched}} \delta + \sum_{j: y_j \text{ unmatched}} \delta}_{\text{gap}}$$

Ex: CTACCG vs. TACATG.

Sol: $M = x_2 - y_1, x_3 - y_2, x_4 - y_3, x_5 - y_4, x_6 - y_6$.

x_1	x_2	x_3	x_4	x_5		x_6
C	T	A	C	C	-	G

	y_1	y_2	y_3	y_4	y_5	y_6
-	T	A	C	A	T	G

Sequence Alignment: Problem Structure

Def. $S[i][j] := \min$ cost of aligning strings $x_1 x_2 \dots x_i$ and $y_1 y_2 \dots y_j$.

- Case 1: S matches x_i - y_j .
 - pay mismatch for x_i - y_j + min cost of aligning two strings $x_1 x_2 \dots x_{i-1}$ and $y_1 y_2 \dots y_{j-1}$
- Case 2a: S leaves x_i unmatched.
 - pay gap for x_i and min cost of aligning $x_1 x_2 \dots x_{i-1}$ and $y_1 y_2 \dots y_j$
- Case 2b: S leaves y_j unmatched.
 - pay gap for y_j and min cost of aligning $x_1 x_2 \dots x_i$ and $y_1 y_2 \dots y_{j-1}$

$$S[i][j] = \begin{cases} j\delta & \text{if } i = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + S[i-1][j-1] \\ \delta + S[i-1][j] \\ \delta + S[i][j-1] \end{cases} & \text{otherwise} \\ i\delta & \text{if } j = 0 \end{cases}$$

Sequence Alignment: Algorithm

```
Sequence-Alignment( $m, n, x_1x_2\dots x_m, y_1y_2\dots y_n, \delta, \alpha$ ) {  
  for  $i = 0$  to  $m$   
     $S[0][i] = i\delta$   
  for  $j = 0$  to  $n$   
     $S[j][0] = j\delta$   
  
  for  $i = 1$  to  $m$   
    for  $j = 1$  to  $n$   
       $S[i][j] = \min(\alpha[x_i, y_j] + S[i-1][j-1],$   
                     $\delta + S[i-1][j],$   
                     $\delta + S[i][j-1])$   
  
  return  $S[m][n]$   
}
```

Analysis. $\Theta(mn)$ time and space.

English words or sentences: $m, n \leq 10$.

Computational biology: $m = n = 100,000$. 10 billions ops OK, but 10GB array?

Sequence Alignment as Weighted Graph

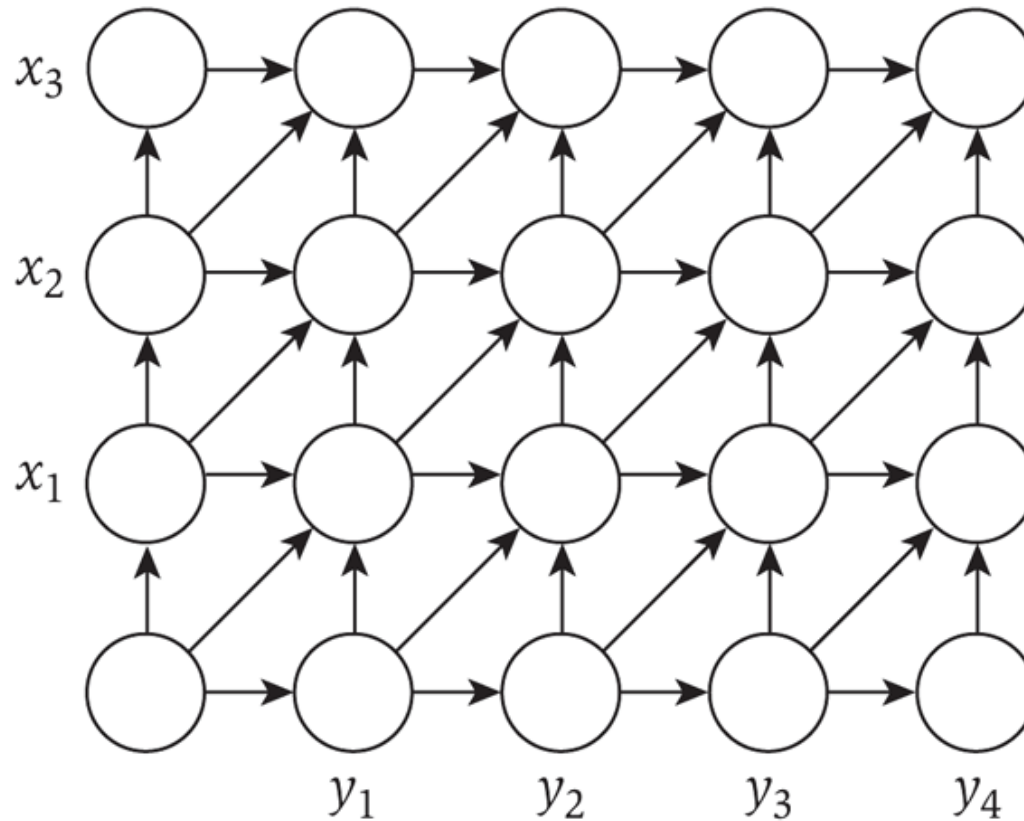


Figure 6.17 A graph-based picture of sequence alignment.

Sequence Alignment: Linear Space

Q. Can we avoid using quadratic **space**?

Easy. Optimal **value** in $O(m)$ space and $O(mn)$ time.

- Compute $S[i][\cdot]$ from $S[i-1][\cdot]$.
- No longer a simple way to recover alignment itself.

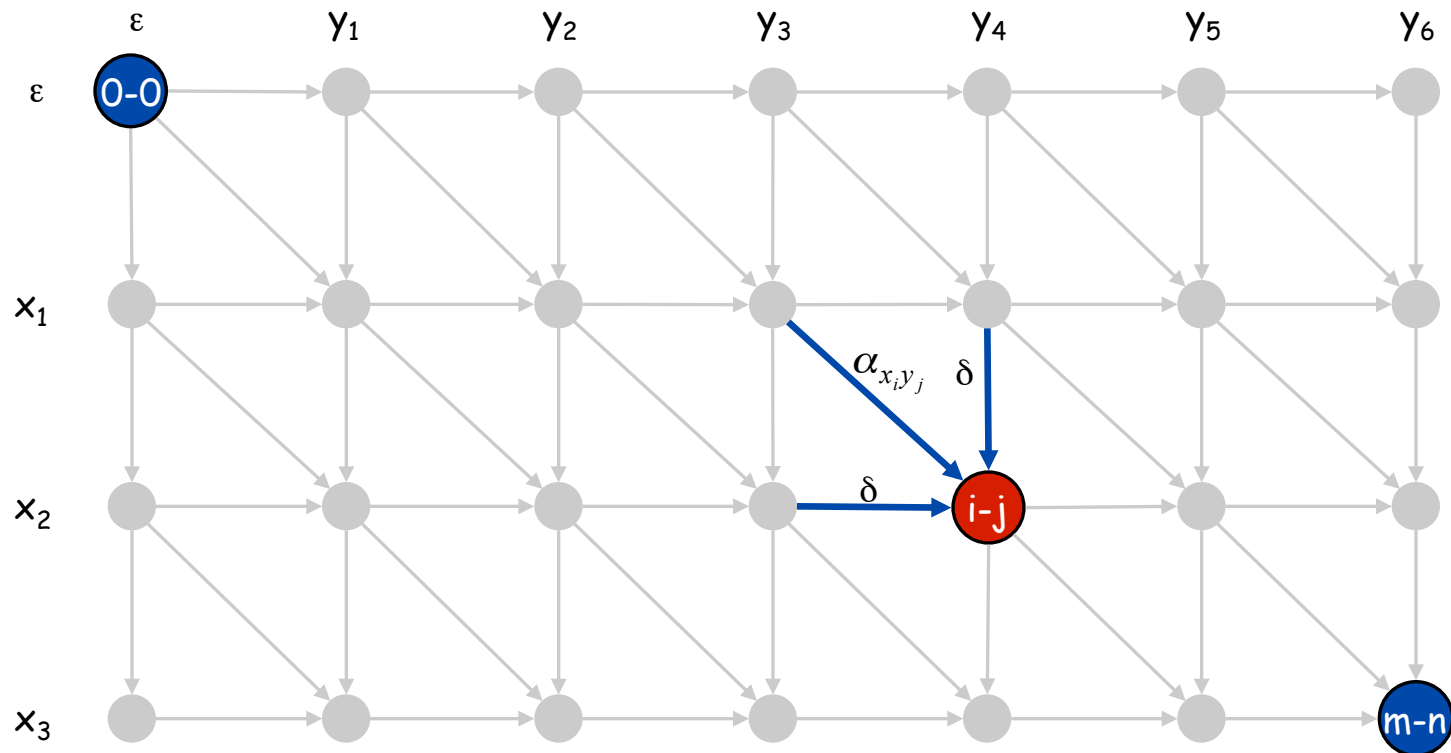
Theorem. [Hirschberg 1975] Optimal **alignment** in $O(m + n)$ space and $O(mn)$ time.

- Clever combination of divide-and-conquer and dynamic programming.

Sequence Alignment: Linear Space

Edit distance graph.

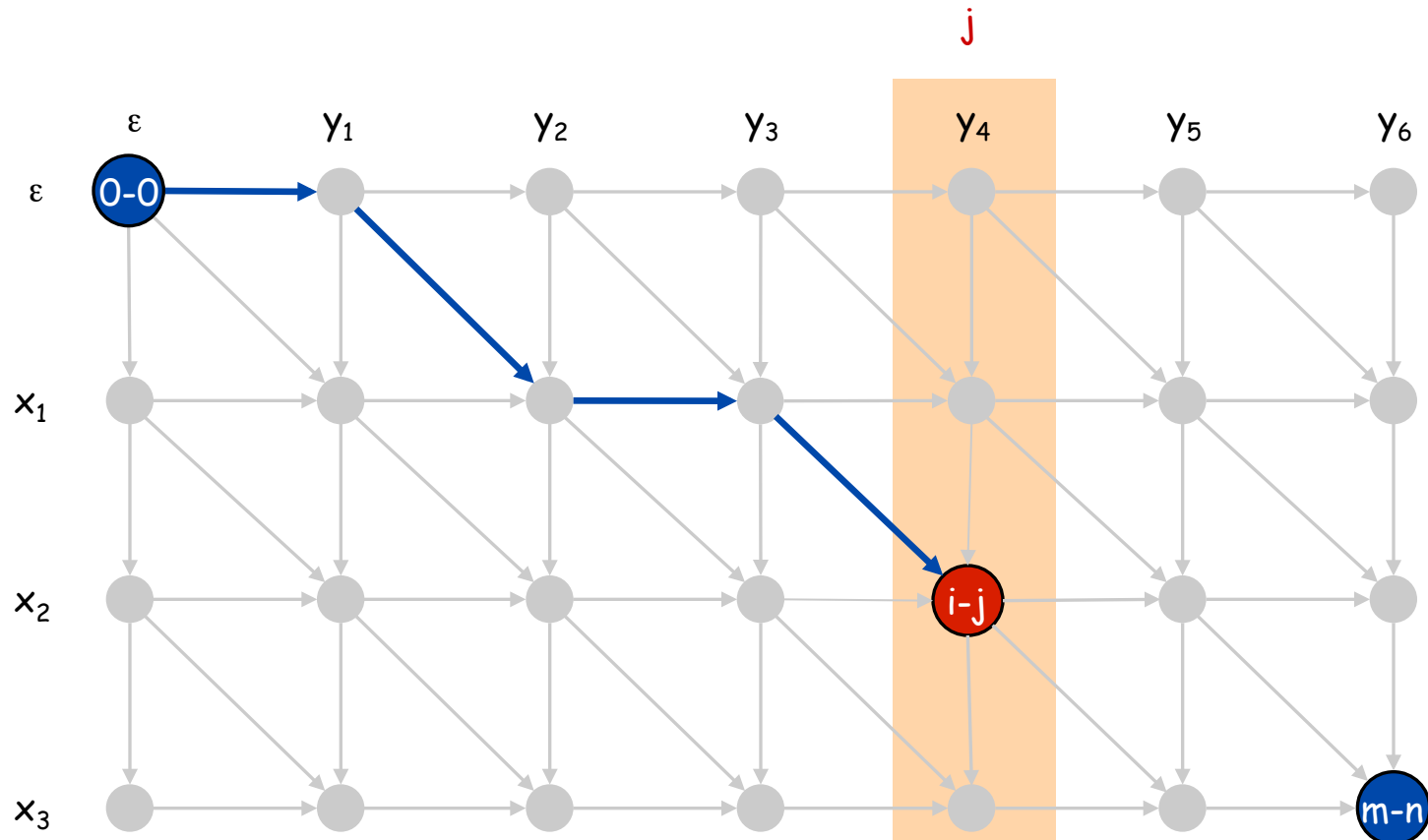
- Let $f(i, j)$ be shortest path from $(0,0)$ to (i, j) .
- Observation: $f(i, j) = S[i][j]$.



Sequence Alignment: Linear Space

Edit distance graph.

- Let $f(i, j)$ be shortest path from $(0,0)$ to (i, j) .
- Can compute $f(\cdot, j)$ for any j in $O(mn)$ time and $O(m + n)$ space.

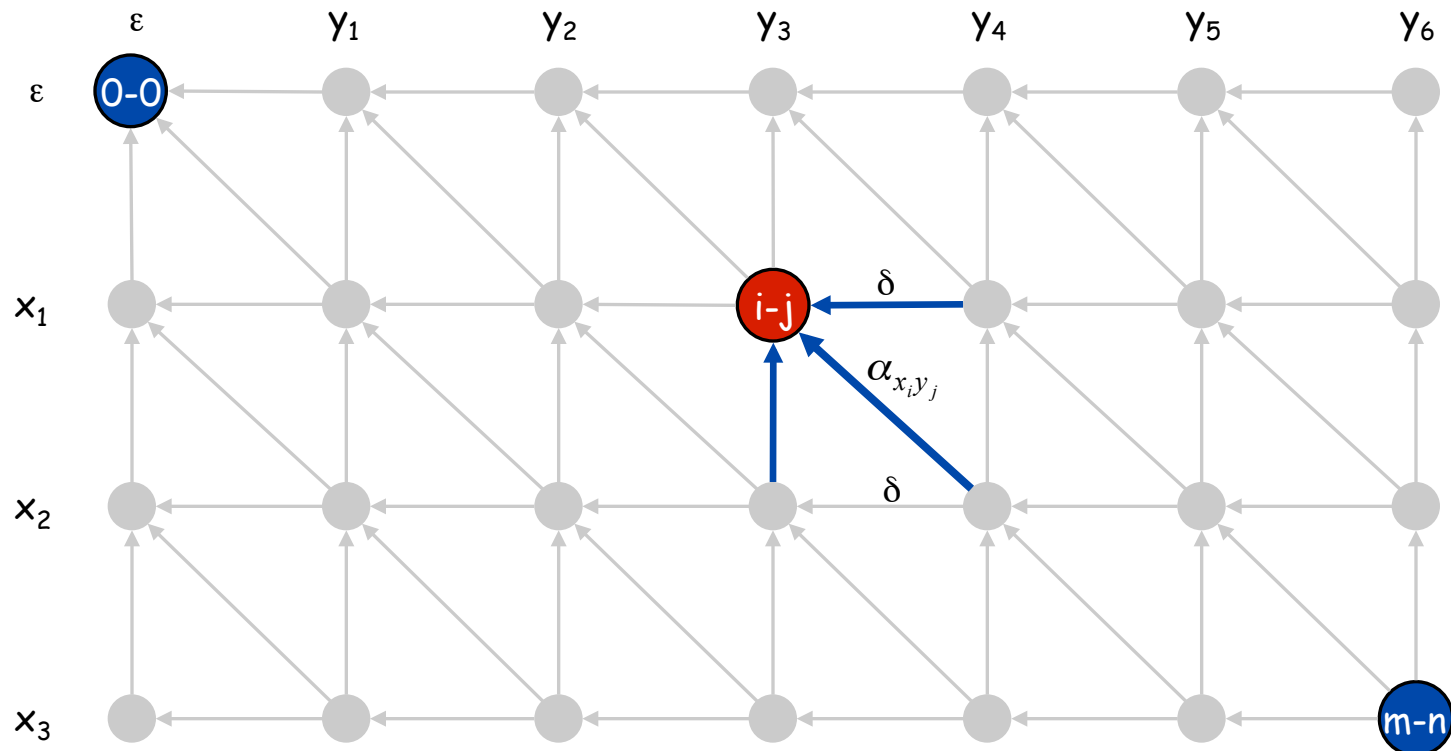


Slides based on Kevin Wayne / Pearson-Addison Wesley

Sequence Alignment: Linear Space

Edit distance graph.

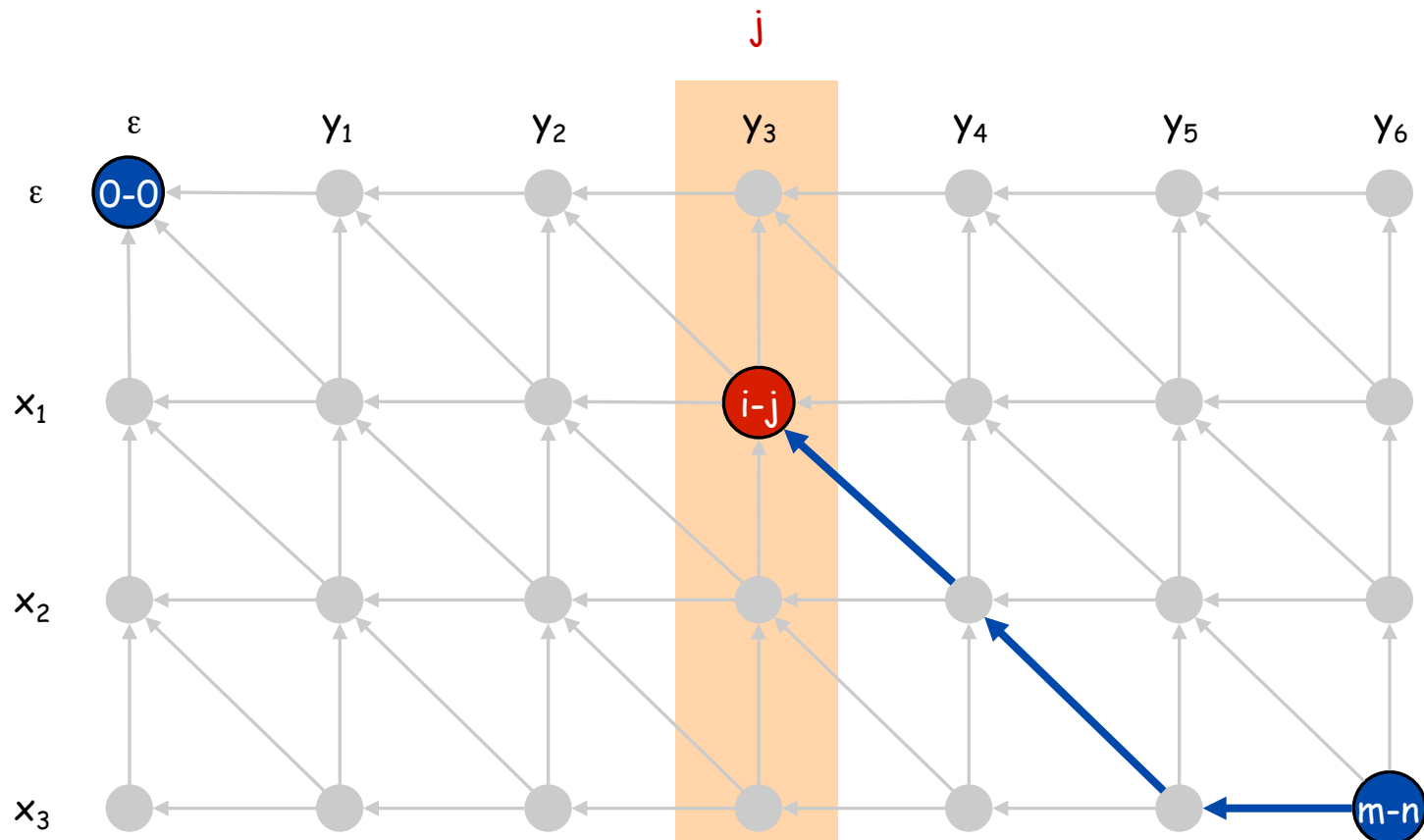
- Let $g(i, j)$ be shortest path from (i, j) to (m, n) .
- Can compute by reversing the edge orientations and inverting the roles of $(0, 0)$ and (m, n)



Sequence Alignment: Linear Space

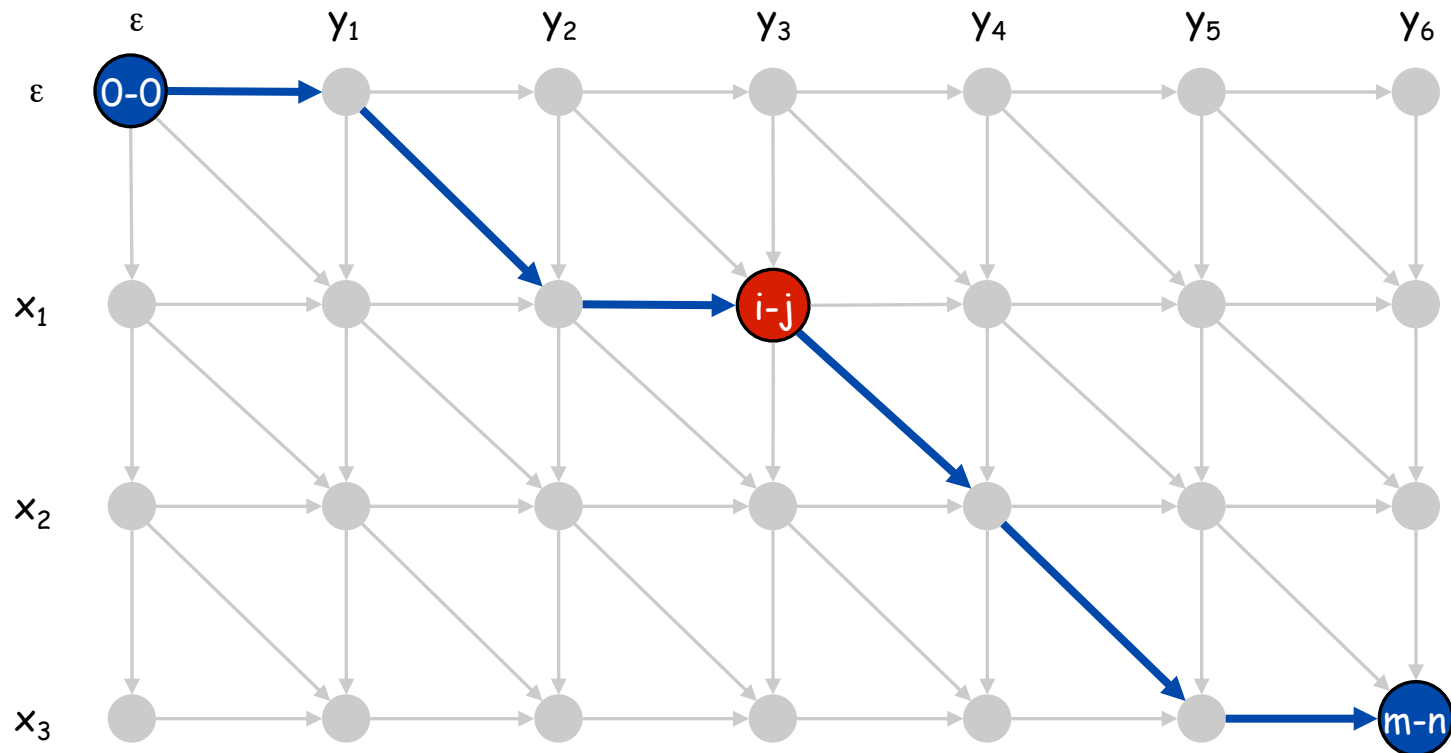
Edit distance graph.

- Let $g(i, j)$ be shortest path from (i, j) to (m, n) .
- Can compute $g(\cdot, j)$ for any j in $O(mn)$ time and $O(m + n)$ space.



Sequence Alignment: Linear Space

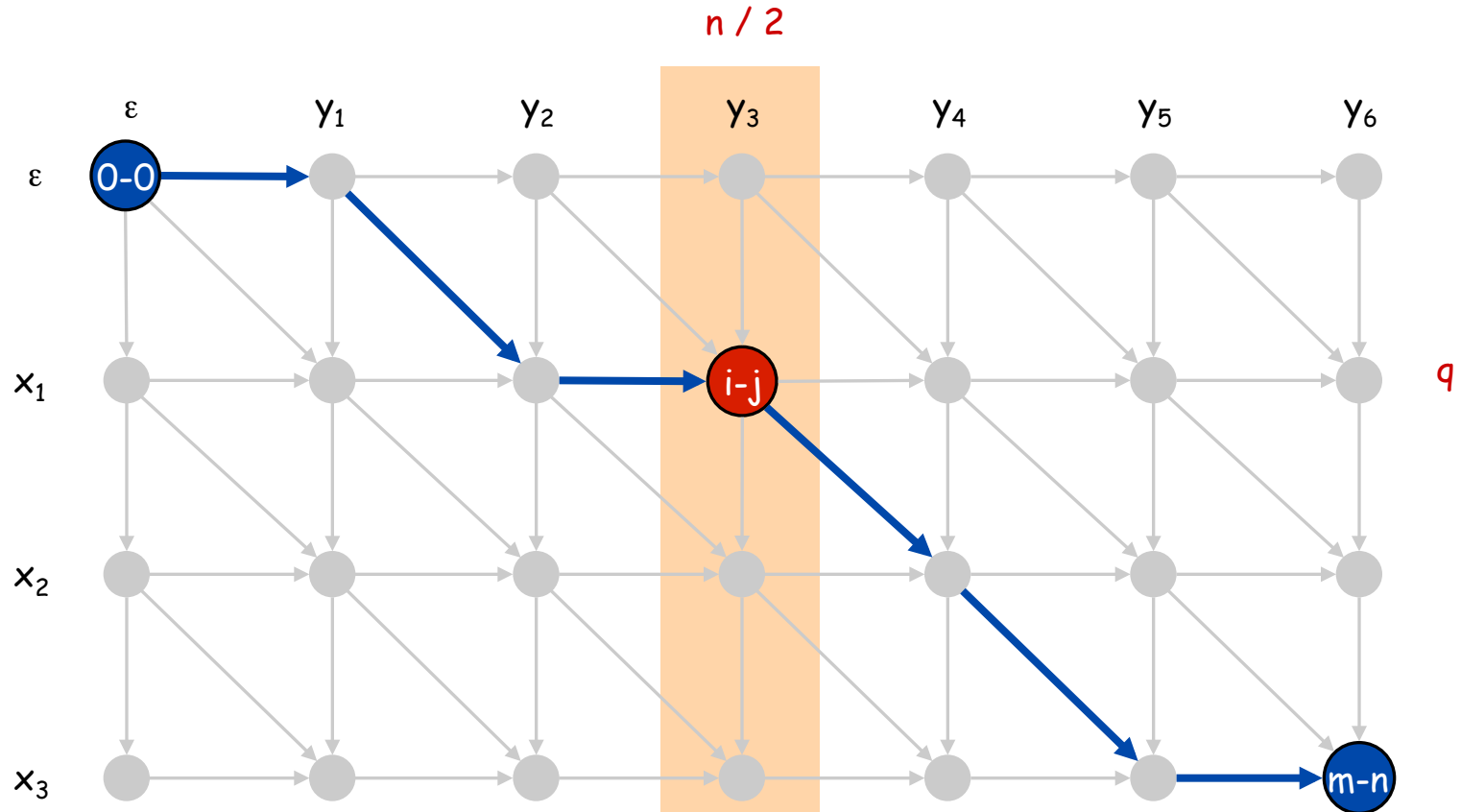
Observation 1. The cost of the shortest path that uses (i, j) is $f(i, j) + g(i, j)$.



Slides based on Kevin Wayne / Pearson-Addison Wesley

Sequence Alignment: Linear Space

Observation 2. let q be an index that minimizes $f(q, n/2) + g(q, n/2)$. Then, the shortest path from $(0, 0)$ to (m, n) uses $(q, n/2)$.

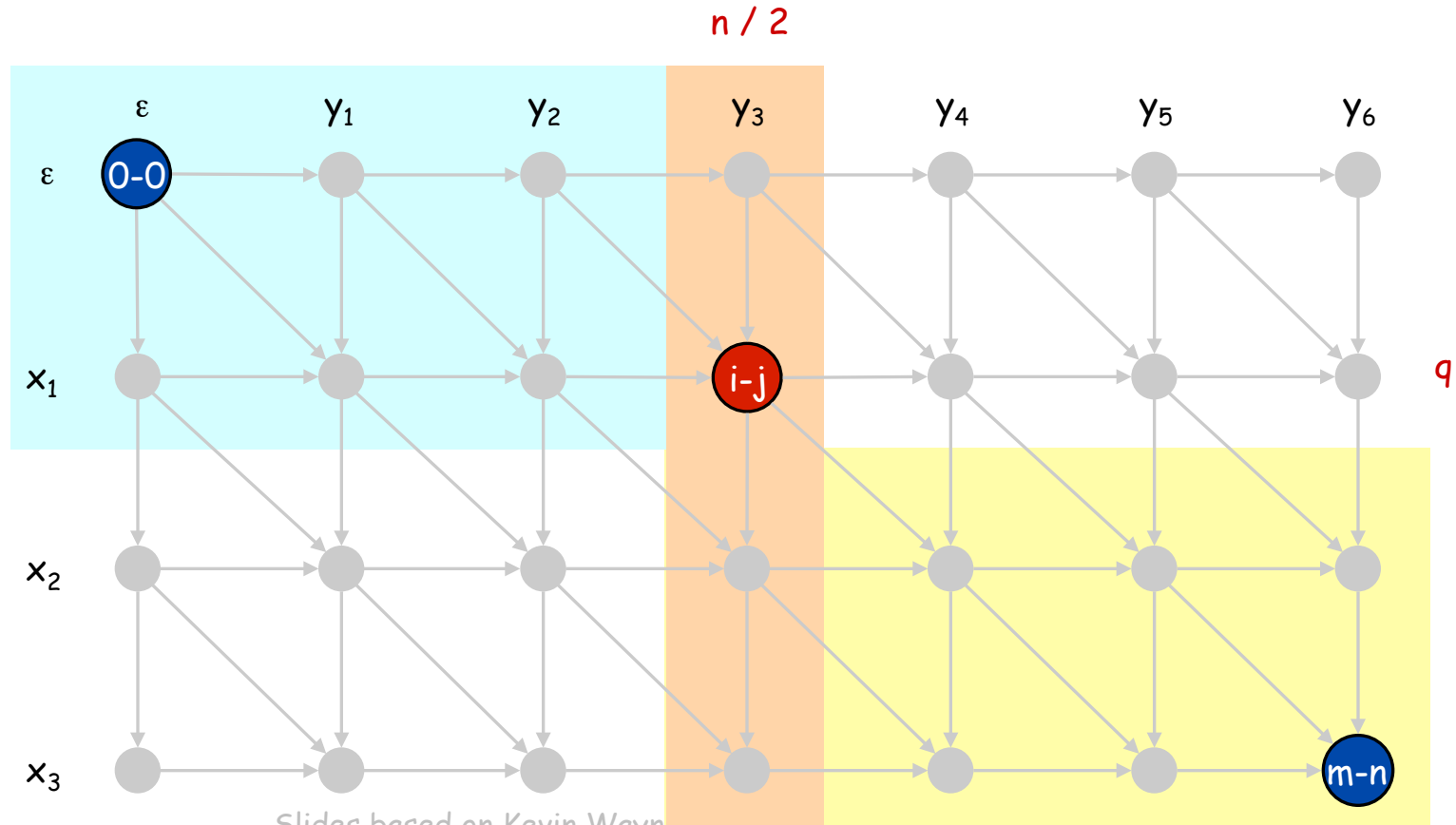


Sequence Alignment: Linear Space

Divide: find index q that minimizes $f(q, n/2) + g(q, n/2)$ using DP.

- Align x_q and $y_{n/2}$.

Conquer: recursively compute optimal alignment in each piece.



Sequence Alignment: Running Time Analysis

Theorem. Let $T(m, n)$ = max running time of algorithm on strings of length m and n . $T(m, n) = O(mn)$.

Pf. (by induction on n)

- $O(mn)$ time to compute $f(\cdot, n/2)$ and $g(\cdot, n/2)$ and find index q .
- $T(q, n/2) + T(m - q, n/2)$ time for two recursive calls.
- Identify constant c so that:

$$T(m, 2) \leq cm$$

$$T(2, n) \leq cn$$

$$T(m, n) \leq cmn + T(q, n/2) + T(m - q, n/2)$$

- Base cases: $m = 2$ or $n = 2$.
- Inductive hypothesis: $T(m, n) \leq 2cmn$.

$$\begin{aligned} T(m, n) &\leq T(q, n/2) + T(m - q, n/2) + cmn \\ &\leq 2cq n/2 + 2c(m - q)n/2 + cmn \\ &= cq n + cmn - cq n + cmn \\ &= 2cmn \end{aligned}$$