6.7 Sequence Alignment in Linear Space

Sequence Alignment

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

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

6 mismatches, 1 gap

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.

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

1 mismatch, 1 gap

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

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

0 mismatches, 3 gaps

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

Sequence Alignment: Linear Space

Q. How to avoid quadratic space when only interested in the value? (1 min)

$$OPT(i, j) = \begin{cases} j\delta & \text{if } i = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if } j = 0 \end{cases}$$

TU Delft

Sequence Alignment: Linear Space

Q. How to avoid quadratic space when only interested in the value?

A. We can calculate the optimal value in O(m + n) space and O(mn) time.

- Compute OPT(i, •) from OPT(i-1, •). Re-use space for "row i-1".
- No longer a simple way to recover alignment itself.

$$OPT(i, j) = \begin{cases} j\delta & \text{if i = 0} \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if j = 0} \end{cases}$$

TU Delft

Sequence Alignment: Linear Space

Q. How to avoid quadratic space when only interested in the value?

A. We can calculate the optimal value in O(m + n) space and O(mn) time.

- Compute OPT(i, •) from OPT(i-1, •). Re-use space for "row i-1".
- No longer a simple way to recover alignment itself.

Q*. How can we still get the solution as well?

TU Delft

Sequence Alignment: Linear Space

Q. How to avoid quadratic space when only interested in the value?

A. We can calculate the optimal value in O(m + n) space and O(mn) time.

- Compute OPT(i, •) from OPT(i-1, •). Re-use space for "row i-1".
- No longer a simple way to recover alignment itself.

Q*. How can we still get the solution as well?

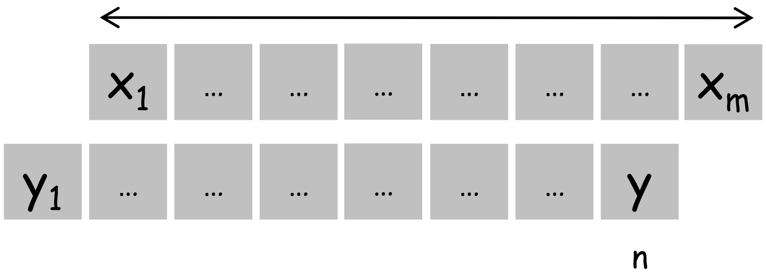
Theorem. [Hirschberg 1975] Optimal alignment in O(m + n) space and O(mn) time.

- Clever combination of divide-and-conquer and dynamic programming.
- Inspired by idea of Savitch from complexity theory.

TU Delft

Sequence Alignment: Divide and conquer

Q. How to apply divide and conquer? How to divide the problem? (1 min)



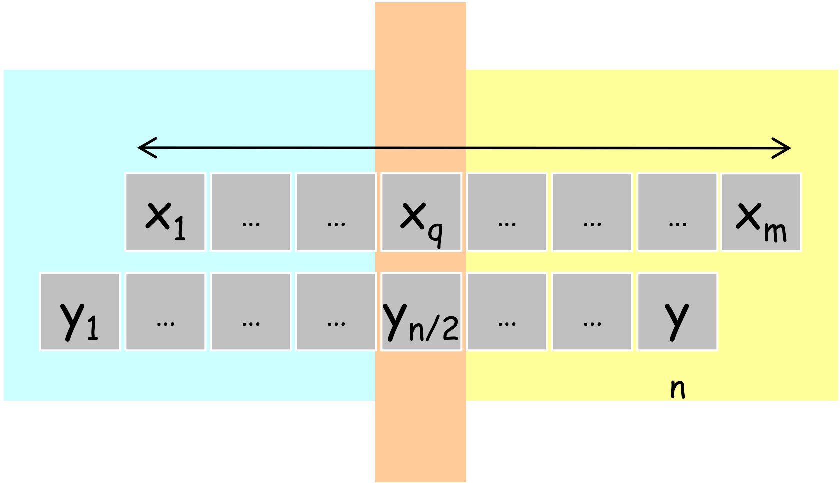
Sequence Alignment: Divide and conquer

A. Cut string y into two halves.
Decide for every index q of x:

potentially requires us to solve m times a sequence alignment problem…

- the optimal alignment up to (q,n/2) and
- the optimal alignment from (q,n/2) to (m,n).

Then, the shortest path from (0, 0) to (m, n) uses the minimum of these.

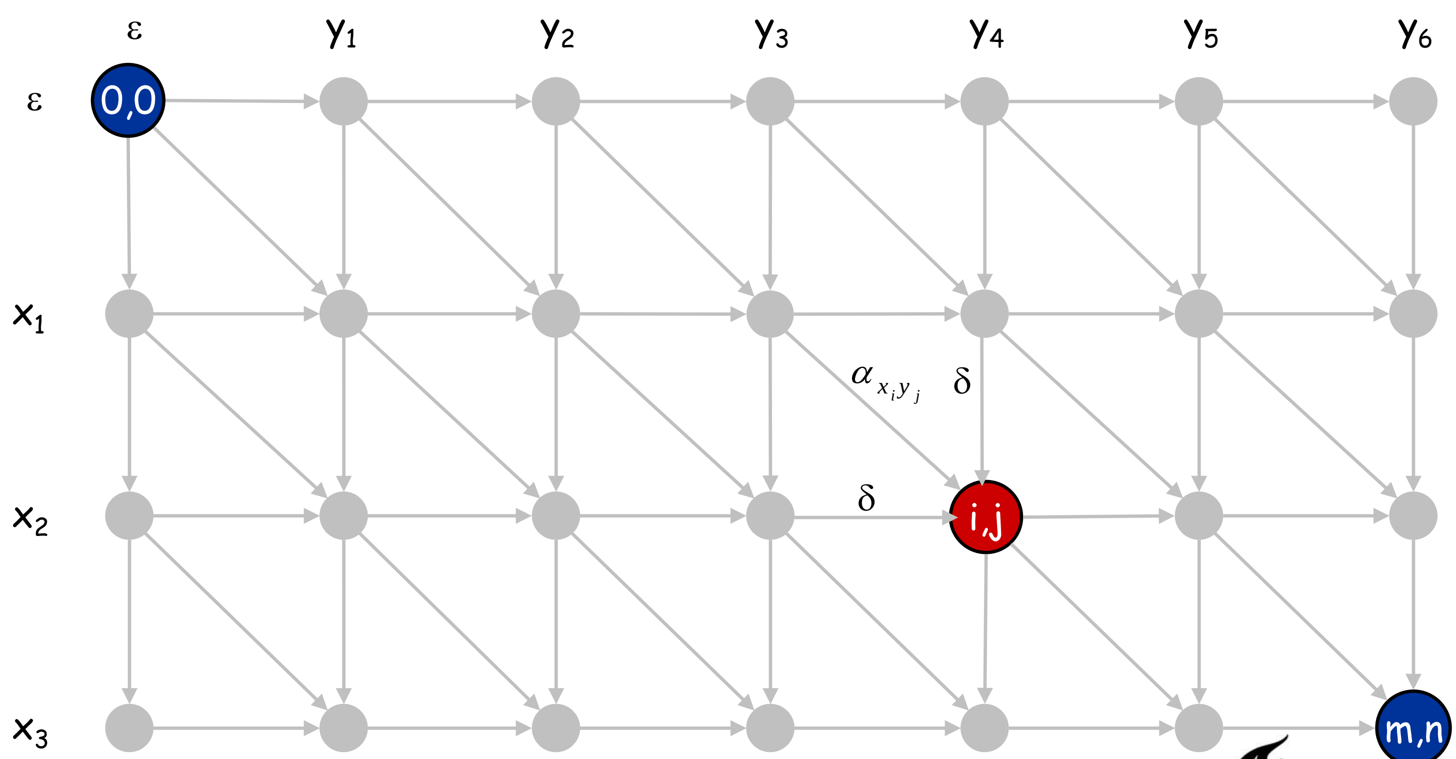


TU Delft

Sequence Alignment: Visualization of the matrix

Edit distance graph.

- Gap penalty δ; mismatch penalty α_{ij}; empty string ε

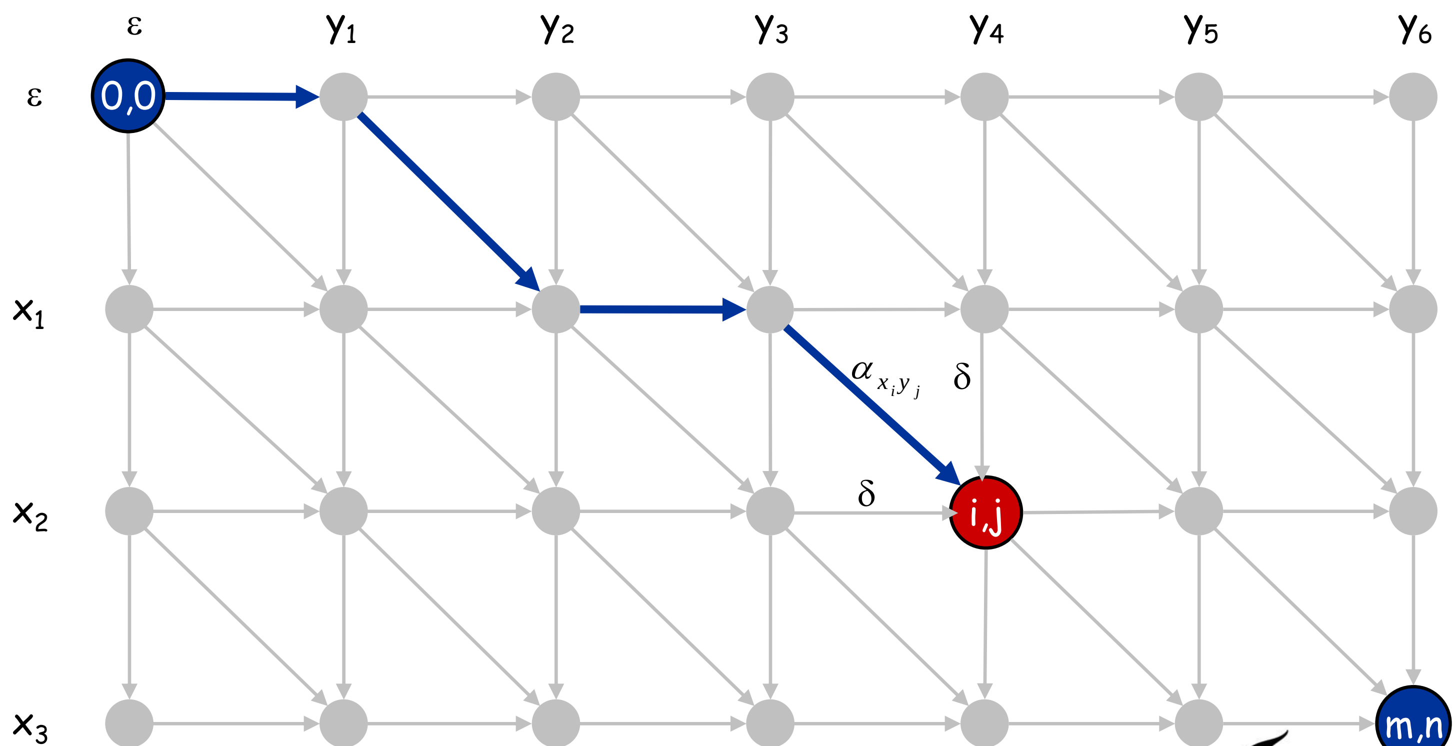


Sequence Alignment: Visualization of the matrix

Edit distance graph.

- Gap penalty δ; mismatch penalty α_{ij}; empty string ε
- Let f(i, j) be shortest path from (0,0) to (i, j).

f(i,j) represents the best way to align $x_1..x_i$ and $y_1..y_j$

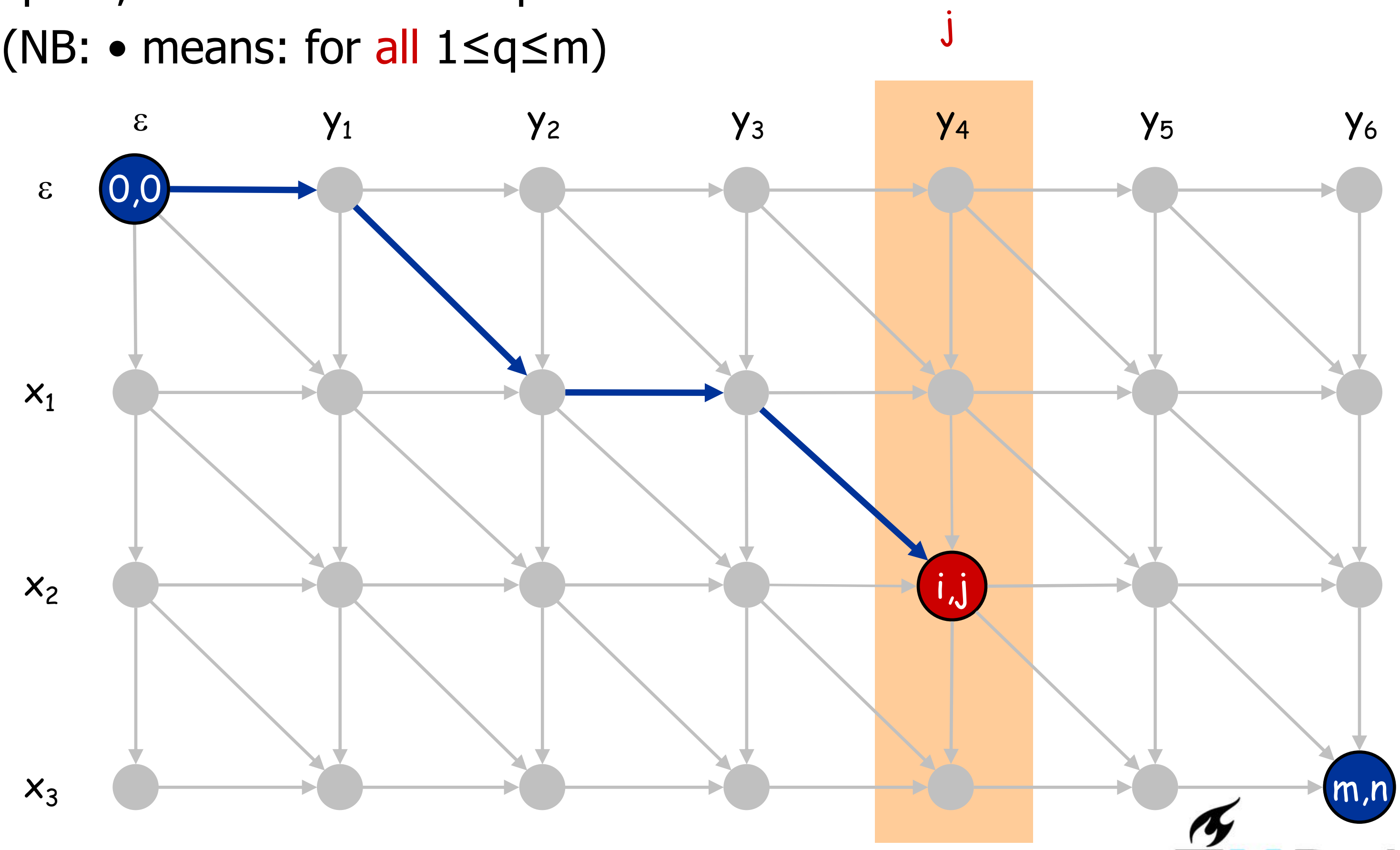


TU Delft

Sequence Alignment: Linear Space

Edit distance graph.

- Let f(i, j) be shortest path from (0,0) to (i, j).
- Can compute length of f (•, j) for any j in O(mn) time and O(m + n) space, because same subproblems are used.
  (NB: • means: for all 1≤q≤m)

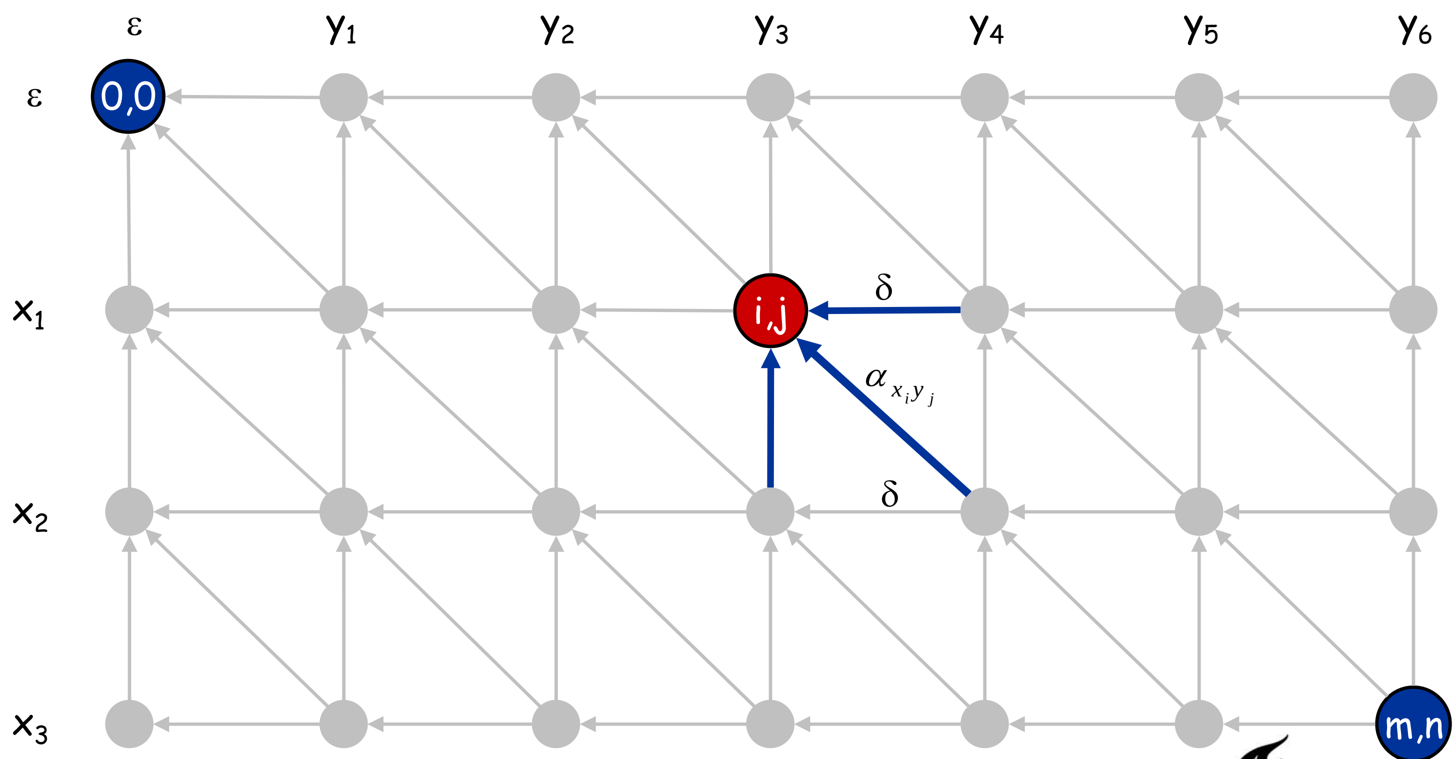


TU Delft

Sequence Alignment: Linear Space

Edit distance graph.

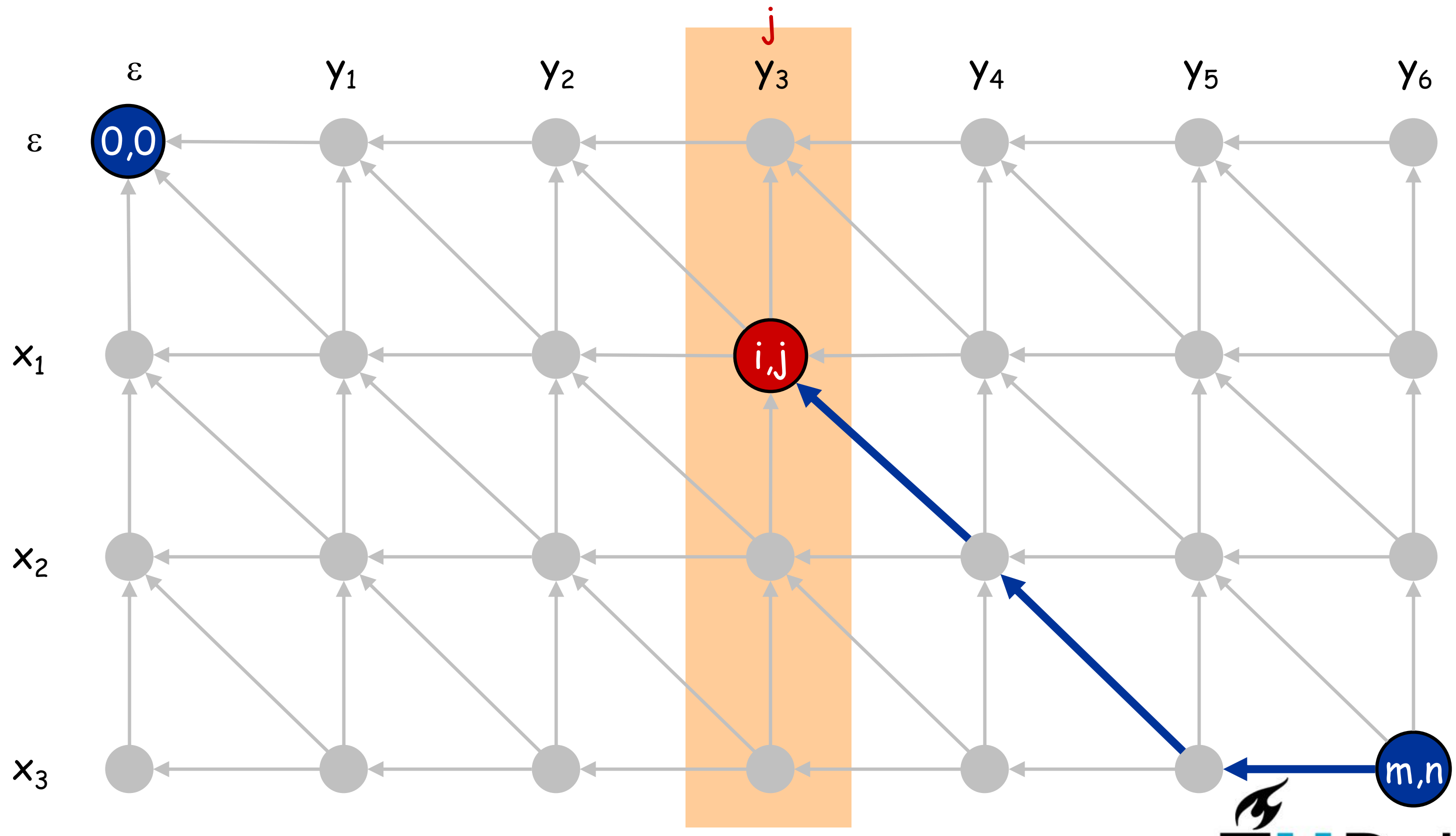
- Let g(i, j) be shortest path from (i, j) to (m, n).
- Can compute g by reversing the edge orientations and inverting the roles of (0, 0) and (m, n)



Sequence Alignment: Linear Space

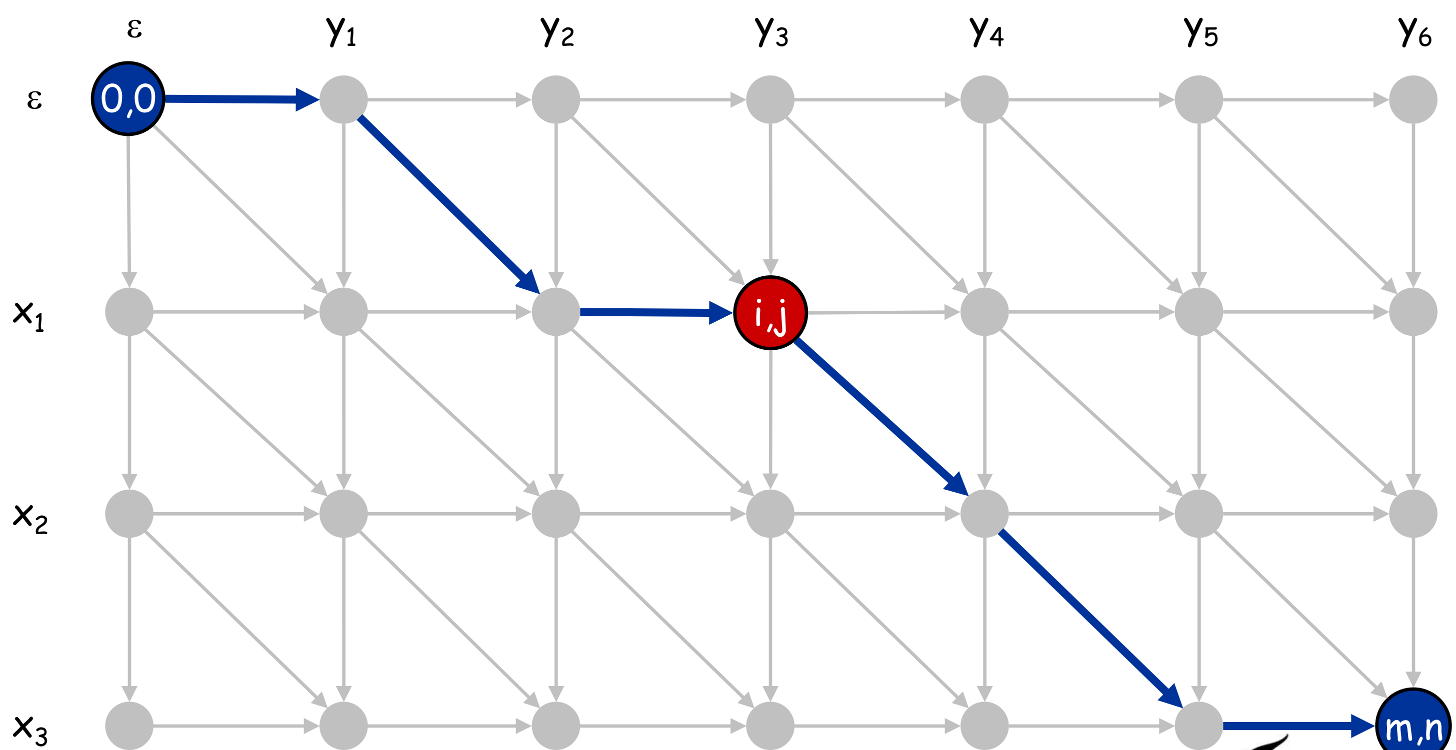
Edit distance graph.

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



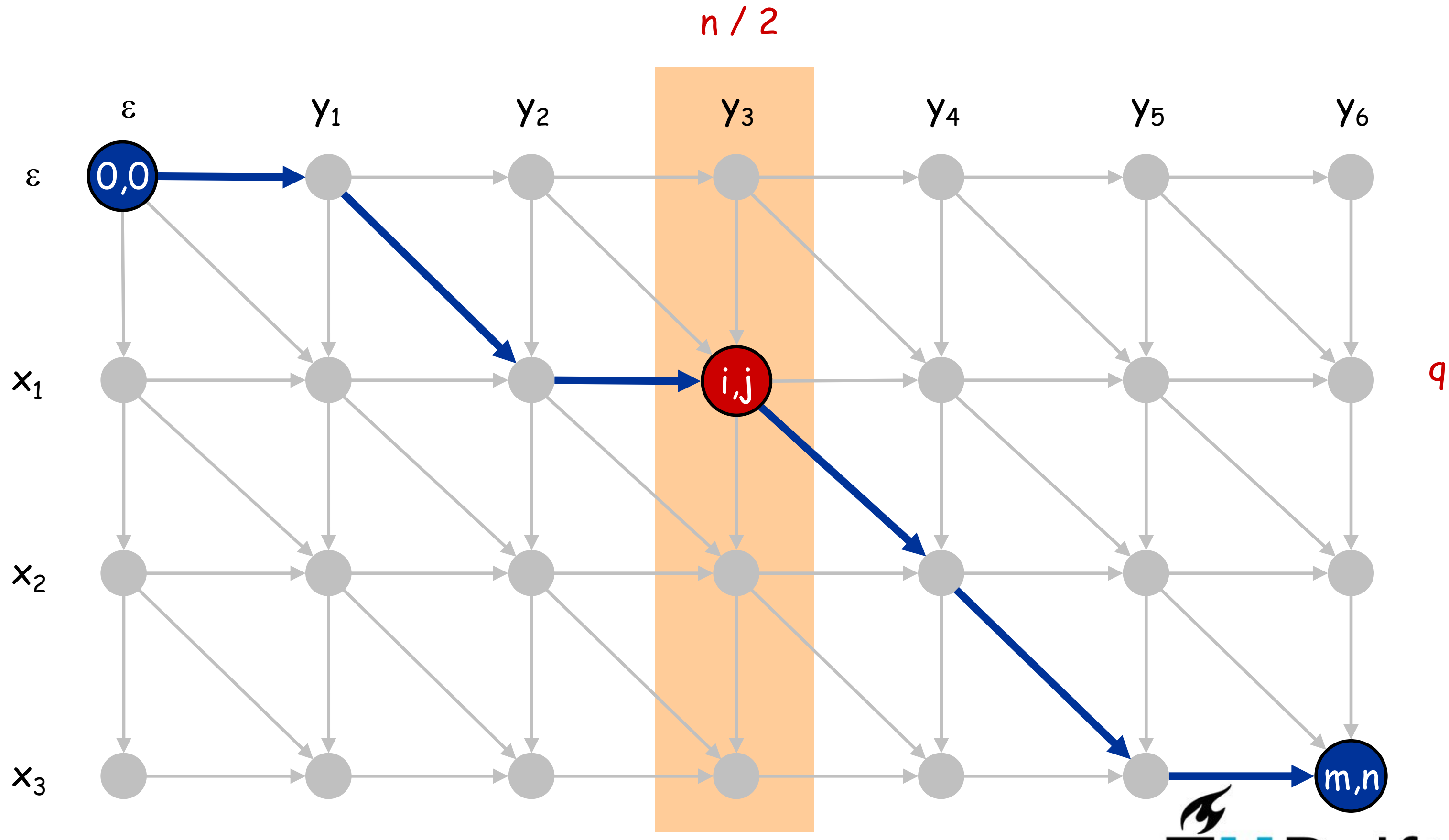
Sequence Alignment: Linear Space

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



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).



TU Delft

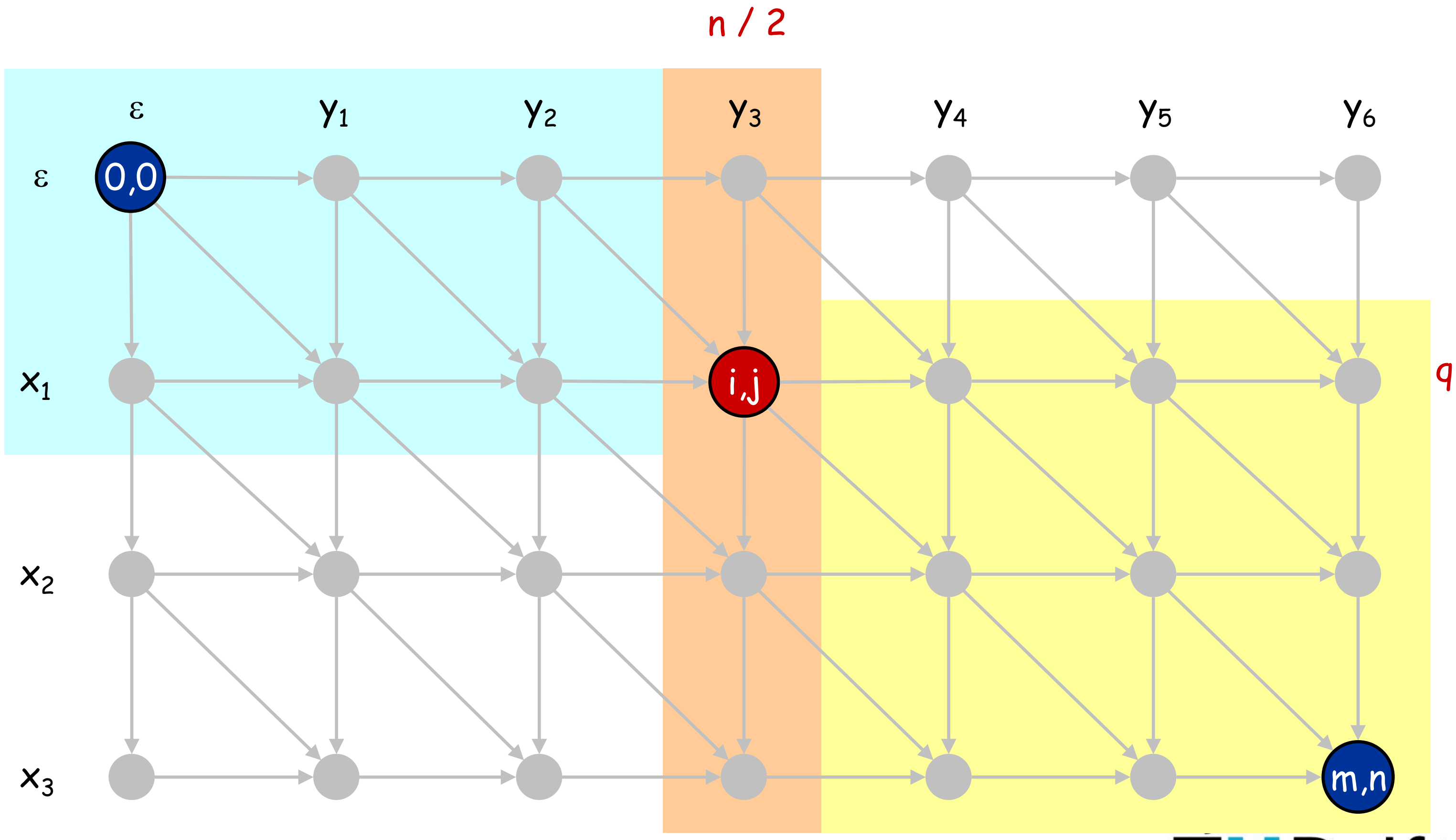
Sequence Alignment: Linear Space

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

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

Conquer: recursively compute optimal alignment in each piece.

Q. What is the running time of this algorithm?



Sequence Alignment: Running Time Analysis Warmup

Q. Let T(m, n) = max running time of algorithm on strings of length at most m and n. Give a tight bound for T.

Pf.

Sequence Alignment: Running Time Analysis Warmup

Q. Let $T(m, n)$ = max running time of algorithm on strings of length at most m and n. Give a tight bound for T.

Pf. $O(mn)$ time to compute $f(\bullet, n/2)$ and $g(\bullet, n/2)$ and find index q.

- then $T(q, n/2) + T(m - q, n/2)$ time for two recursive calls.

TU Delft

Sequence Alignment: Running Time Analysis Warmup

Q. Let T(m, n) = max running time of algorithm on strings of length at most m and n. Give a tight bound for T.

Pf. O(mn) time to compute f(•, n/2) and g(•, n/2) and find index q.

- then T(q, n/2) + T(m - q, n/2) time for two recursive calls.
- $T(m,n) \leq 2T(m, n/2) + O(mn)$ does not fit Master theorem

Sequence Alignment: Running Time Analysis Warmup

Q. Let T(m, n) = max running time of algorithm on strings of length at most m and n. Give a tight bound for T.

Pf. O(mn) time to compute f(•, n/2) and g(•, n/2) and find index q.

- then T(q, n/2) + T(m - q, n/2) time for two recursive calls.
- $T(m,n) \le 2T(m, n/2) + O(mn)$ does not fit Master theorem,
- first try for m=n: $T(n) \le 2T(n/2) + O(n^2)$

Q. Use the master method to solve this recurrence relation.

TU Delft

Sequence Alignment: Running Time Analysis Warmup

Q. Let T(m, n) = max running time of algorithm on strings of length at most m and n. Give a tight bound for T.

Pf. O(mn) time to compute f(•, n/2) and g(•, n/2) and find index q.

- then T(q, n/2) + T(m - q, n/2) time for two recursive calls.
- $T(m,n) \leq 2T(m, n/2) + O(mn)$ does not fit Master theorem,
- first try for m=n: $T(n) \leq 2T(n/2) + O(n^2)$

Q. Use the master method to solve this recurrence relation.

$$a = 2,\ b = 2; \qquad n^{\log_b a} = n^{\log_2 2} = n$$

$$f(n) = \Omega(n^{1+\varepsilon})$$

- So this is case?

Sequence Alignment: Running Time Analysis Warmup

Q. Let T(m, n) = max running time of algorithm on strings of length at most m and n. Give a tight bound for T.

Pf. O(mn) time to compute f(•, n/2) and g(•, n/2) and find index q.

- then T(q, n/2) + T(m - q, n/2) time for two recursive calls.
- $T(m,n) \le 2T(m, n/2) + O(mn)$ does not fit Master theorem,
- first try for m=n: $T(n) \le 2T(n/2) + O(n^2)$

Q. Use the master method to solve this recurrence relation.

$$a = 2,\ b = 2; \qquad n^{\log_b a} = n^{\log_2 2} = n$$

$$f(n) = \Omega(n^{1+\varepsilon})$$

- So this is case 3. Check regularity condition: $2f(n/2) = 2(n/2)^2 \le cn^2$; e.g. for c=1/2, and thus: $T(n) = \Theta(n^2)$

Q. How to solve this for any m?

$$T(m,n) \le T(q, n/2) + T(m-q, n/2) + O(mn)$$

TUDelft

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).

$$T(m,n) \le T(q, n/2) + T(m-q, n/2) + O(mn)$$

Pf. By induction on n we show that T(m, n) $\le$ 2cmn

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).

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

Pf. By induction on n we show that T(m, n) ≤ 2cmn

- Choose constant c so that:

$$\begin{array}{lcl} T(m,\ 2) & \leq & cm \\ T(2,\ n) & \leq & cn \\ T(m,\ n) & \leq & cmn + T(q,\ n/2) + T(m-q,\ n/2) \end{array}$$

- **Base cases:** m = 2 or n = 2: immediate from def. of c.
- **Inductive hypothesis:** T(m′, n′) ≤ 2cm′n′ for m′<m and n′<n.
- **Step:**

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).

$$T(m,n) \le T(q, n/2) + T(m-q, n/2) + O(mn)$$

Pf. By induction on n we show that T(m, n) ≤ 2cmn

- Choose constant c so that:

$$\begin{aligned} T(m,\ 2) &\le cm \\ T(2,\ n) &\le cn \\ T(m,\ n) &\le cmn + T(q,\ n/2) + T(m-q,\ n/2) \end{aligned}$$

- **Base cases:** m = 2 or n = 2.
- **Inductive hypothesis:** T(m′, n′) ≤ 2cm′n′ for m′<m and n′<n.
- **Step:**

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

← Q. What can we use here?

TU Delft

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).

$$T(m,n) \le T(q, n/2) + T(m-q, n/2) + O(mn)$$

Pf. By induction on n we show that T(m, n) ≤ 2cmn

- Choose constant c so that:

$$\begin{aligned} T(m, 2) &\le cm \\ T(2, n) &\le cn \\ T(m, n) &\le cmn + T(q, n/2) + T(m-q, n/2) \end{aligned}$$

- **Base cases:** m = 2 or n = 2.
- **Inductive hypothesis:** T(m′, n′) ≤ 2cm′n′ for m′<m and n′<n.
- **Step:**

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

← use inductive hypothesis

This proof can be found on page 289-290

TU Delft