CS 381 – FALL 2019

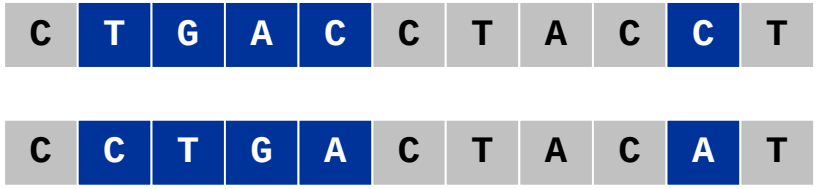
Week 8.2, Wed, Oct 9

No PSOs this week (October Break)
Thursday/Friday PSOs are now ``office hours"
Homework 4 Due: October 14th @ 11:59PM (Gradescope)

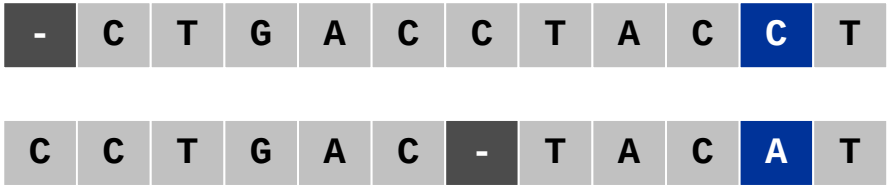
Edit Distance

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

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



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



$2\delta + \alpha_{CA}$

Applications.

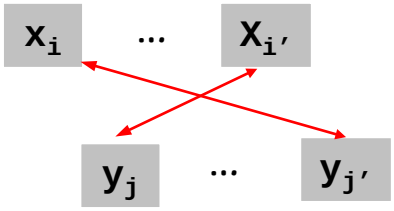
- Basis for Unix diff.
- Speech recognition.
- Computational biology.

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 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.

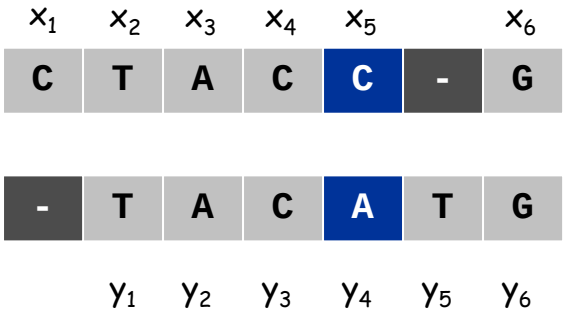
Def. The pair x_i-y_j and $x_{i'}$-$y_{j'}$ cross if $i < i'$, but $j > j'$.



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

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.



Sequence Alignment: Problem Structure

Def. OPT(i, j) = min cost of aligning strings $x_1\, x_2 \ldots x_i$ and $y_1\, y_2 \ldots y_j$.

Recurrence:

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

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

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

Computational biology: $m = n = 1{,}000{,}000$.

1 trillion ops OK, but 1TB array?

Clicker Question

Suppose that x="CT" and y="TA" and

$$\alpha_{x_i,y_j} = \begin{cases} 3\delta & \text{if } x_i \neq y_j \\ 0 & otherwise \end{cases}$$

What are the missing values of x1 and x2 below?

A. $x1 = 3\delta$ and $x2 = 4\delta$
B. $x1 = 3\delta$ and $x2 = 3\delta$
C. $x1 = 2\delta$ and $x2 = 5\delta$
D. $x1 = 2\delta$ and $x2 = 2\delta$
E. $x1 = \delta$ and $x2 = 2\delta$

	j=0	j=1	j=2
i=0	0	δ	2δ
i=1	δ	2δ	3δ
i=2	2δ	$x1 = ??$	$x2 = ??$

OPT(i,j)

x_1	x_2
C	T
T	A
y_1	y_2

Clicker Question

Suppose that x="CT" and y="TA" and

$$\alpha_{x_i,y_j} = \begin{cases} 3\delta & \text{if } x_i \neq y_j \\ 0 & otherwise \end{cases}$$

What are the missing values of x1 and x2 below?

A. $x1 = 3\delta$ and $x2 = 4\delta$
B. $x1 = 3\delta$ and $x2 = 3\delta$
C. $x1 = 2\delta$ and $x2 = 5\delta$
D. $x1 = 2\delta$ and $x2 = 2\delta$
E. $x1 = \delta$ and $x2 = 2\delta$

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

	j=0	j=1	j=2
i=0	0	δ	2δ
i=1	δ	2δ	3δ
i=2	2δ	$x1 = ??$	$x2 = ??$

OPT(i,j)

x_1	x_2	
C	T	-

-	T	A

y_1 y_2

6.7 Sequence Alignment in Linear Space

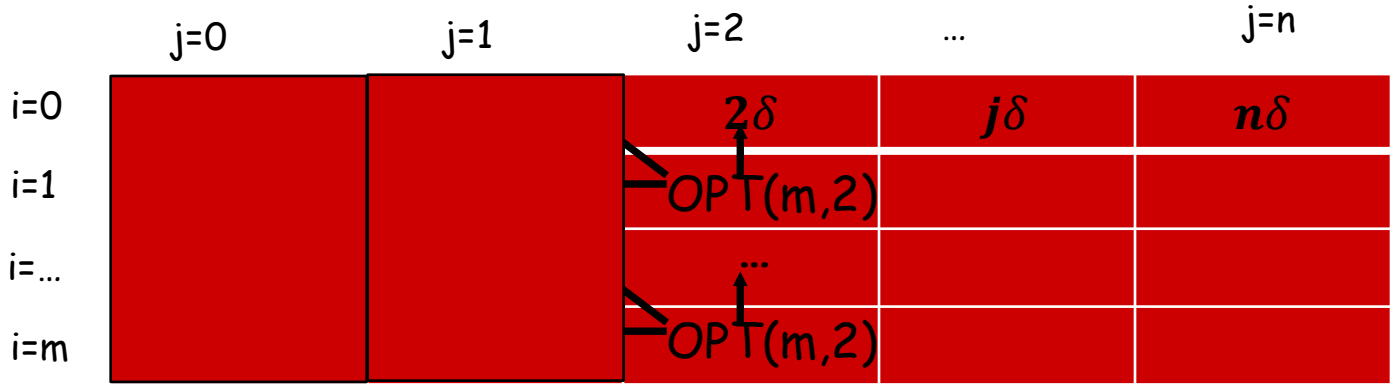
Sequence Alignment: Linear Space

Q. Can we avoid using quadratic space?

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

- Compute $OPT(i, \cdot)$ from $OPT(i-1, \cdot)$.
- No longer a simple way to recover alignment itself.

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



Sequence Alignment: Linear Space

Q. Can we avoid using quadratic space?

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

- Compute $OPT(i, \cdot)$ from $OPT(i-1, \cdot)$.
- 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.
- Inspired by idea of Savitch from complexity theory.

Sequence Alignment: Linear Space

Recall. OPT(i, j) = min cost of aligning strings $x_1\, x_2 \ldots x_i$ and $y_1\, y_2 \ldots y_j$.

Def. $OPT_{rev}(i, j)$ = min cost of aligning strings $x_{i+1}\, x_{i+2} \ldots x_m$ and $y_{j+1}\, y_{j+2} \ldots y_n$.

Symmetry: Can develop recurrence for $OPT_{rev}(i, j)$

- Computing OPT(i, j) requires OPT(i,j-1) and OPT(i-1,j)
- Computing $OPT_{rev}(i, j)$ requires $OPT_{rev}(i+1, j)$ and $OPT_{rev}(i, j+1)$

Observation: For any j< n we have

$$OPT(m, n) = \min_{0 \le i \le m} \{OPT(i, j) + OPT_{rev}(i, j)\}$$

Cost to aligning strings $x_1\, x_2 \ldots x_i$ and $y_1\, y_2 \ldots y_j$

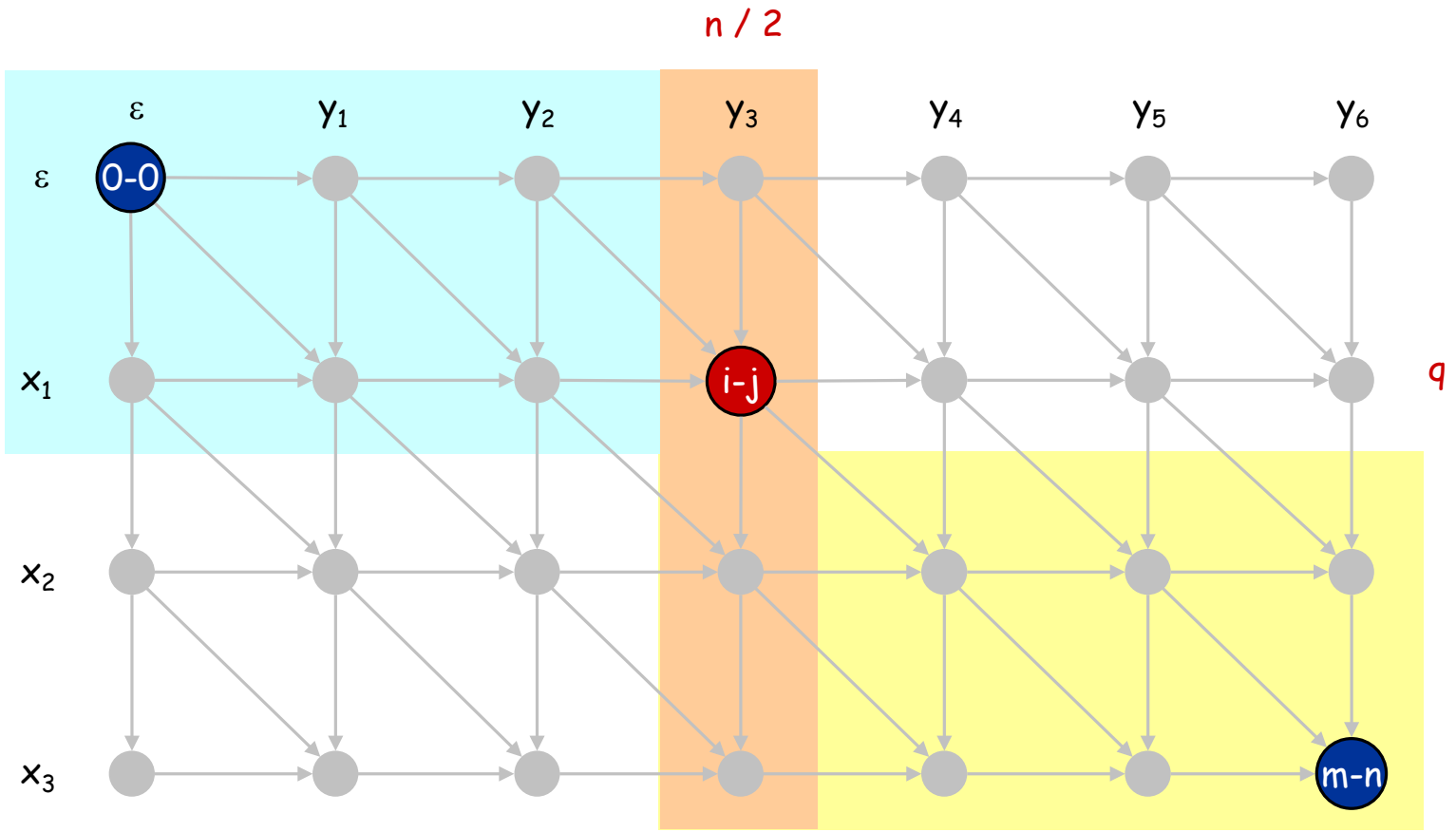
Cost to align remainder $x_{i+1}\, x_{i+2} \ldots x_m$ and $y_{j+1}\, y_{j+2} \ldots y_n$

Sequence Alignment: Linear Space

Divide: find index q that minimizes OPT (q, n/2) + OPT_{rev} (q, n/2) using DP.

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

Conquer: recursively compute optimal alignment in each piece.

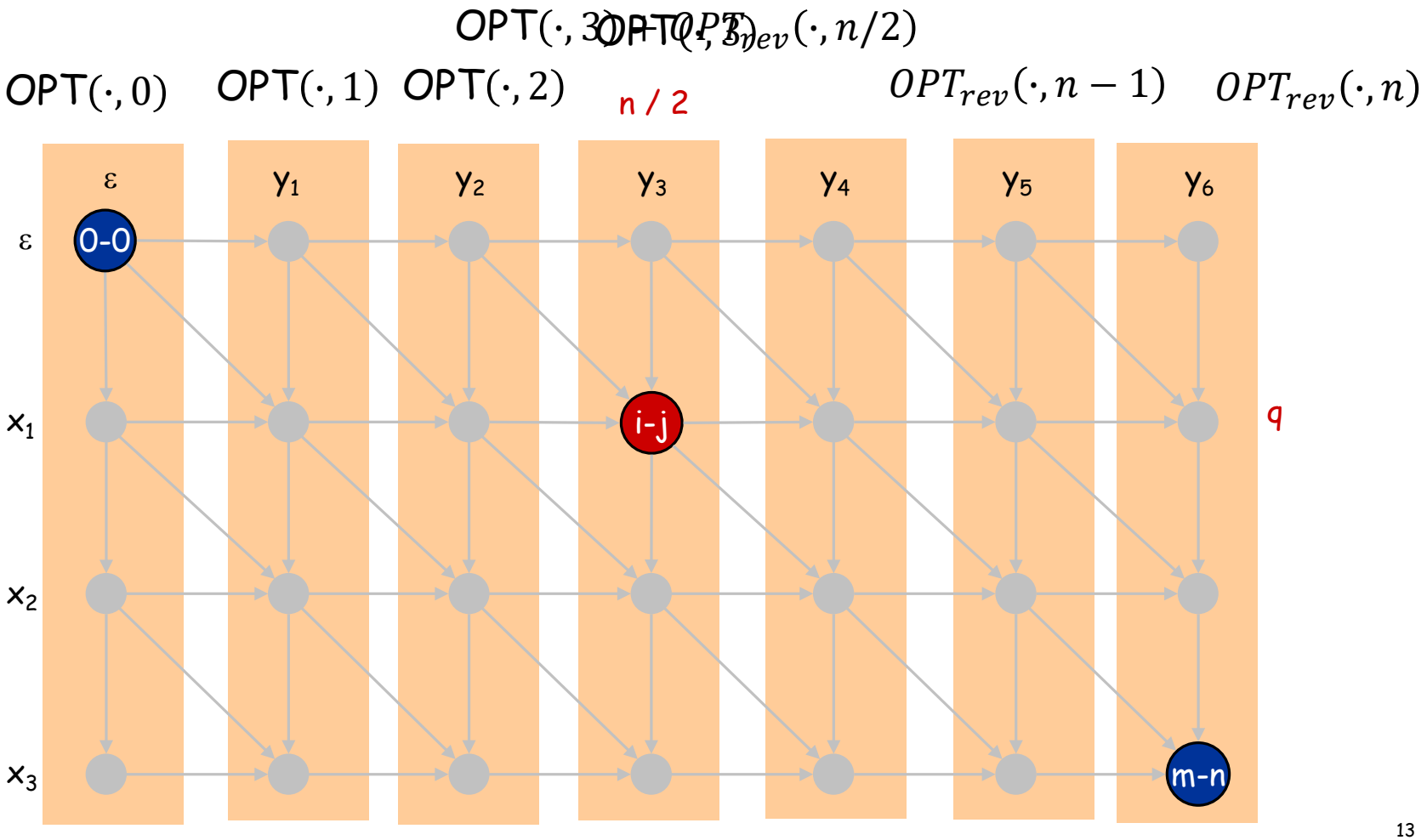


Sequence Alignment: Linear Space

Divide: find index q that minimizes OPT (q, n/2) + OPT_{rev} (q, n/2) using DP.

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

Conquer: recursively compute optimal alignment in each piece.



Sequence Alignment: Linear Space

Recall. OPT(i, j) = min cost of aligning strings $x_1\ x_2\ \ldots\ x_i$ and $y_1\ y_2\ \ldots\ y_j$.

Def. $OPT_{rev}(i, j)$ = min cost of aligning strings $x_{i+1}\ x_{i+2}\ \ldots\ x_m$ and $y_{j+1}\ y_{j+2}\ \ldots\ y_n$.

Symmetry: Can develop recurrence for $OPT_{rev}(i, j)$

- Computing OPT(i, j) requires OPT(i,j-1) and OPT(i-1,j)
- Computing $OPT_{rev}(i, j)$ requires $OPT_{rev}(i+1, j)$ and $OPT_{rev}(i, j+1)$

Observation: For any j< n we have

$$OPT(m,n) = \min_{0 \le i \le m} \{OPT(i,j) + OPT_{rev}(i,j)\}$$

Divide: Set j = n/2 and compute rows $OPT(\cdot, j)$ and $OPT_{rev}(\cdot\ , j)$

- Find q minimizing $OPT(q,j) + OPT_{rev}(q,j)$, free memory and recurse
- Time: O(mn)
- Space: O(m+n)
  - Why? Only require row $OPT(\cdot, j-1)$ to compute next row $OPT(\cdot, j)$

Sequence Alignment: Linear Space

Recall. OPT(i, j) = min cost of aligning strings $x_1\, x_2 \ldots x_i$ and $y_1\, y_2 \ldots y_j$.

Def. $OPT_{rev}(i, j)$ = min cost of aligning strings $x_{i+1}\, x_{i+2} \ldots x_n$ and $y_{j+1}\, y_{j+2} \ldots y_n$.

Symmetry: Can develop recurrence for $OPT_{rev}(i, j)$

- Computing OPT(i, j) requires OPT(i,j-1) and OPT(i-1,j)
- Computing $OPT_{rev}(i, j)$ requires $OPT_{rev}(i+1, j)$ and $OPT_{rev}(i, j+1)$

Observation: For any j< m we have

$$OPT(m, n) = \min_{0 \le i \le m} \{OPT(i, j) + OPT_{rev}(i, j)\}$$

Conquer/Merge: Set j = n/2 and compute rows $OPT(\cdot, j)$ and $OPT_{rev}(\cdot\, , j)$

- Find q minimizing $OPT(q, j) + OPT_{rev}(q, j)$, free memory and recurse
- **Sub-problem 1:** Align $x_1\, x_2 \ldots x_q$ and $y_1\, y_2 \ldots y_{n/2}$
- **Sub-problem 2:** Align $x_{q+1}\, x_{i+2} \ldots x_m$ and $y_{n/2+1}\, y_{n/2+2} \ldots y_n$

Sequence Alignment: Running Time Analysis Warmup

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

$$T(m, n) \leq 2T(m, n/2) + O(mn) \Rightarrow T(m, n) = O(mn \log n)$$

Remark. Analysis is not tight because two sub-problems are of size (q, n/2) and (m - q, n/2). In next slide, we save log n factor.

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 $OPT(\cdot, n/2)$ and $OPT_{rev}(\cdot, n/2)$ and find index q.
- $T(q, n/2) + T(m - q, n/2)$ time for two recursive calls.
- Choose constant c so that:

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

- Base cases: m = 2 or n = 2.
- Inductive hypothesis: P(n-1)="$T(m, n') \leq 2cmn'$ for all $n' < n$ and all $m > 0$"

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