Mitchell Denen
Sowmya
Shahin Safarov

Dynamic programming.

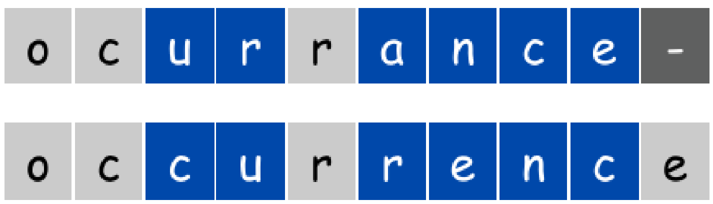
Break up a problem into a series of overlapping sub-problems, and build up solutions to larger and larger sub-problems.

Sequence Alignment

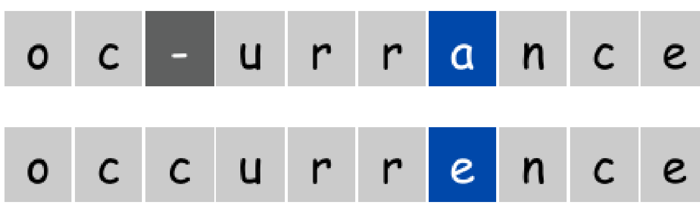
String Similarity

How similar are two strings?

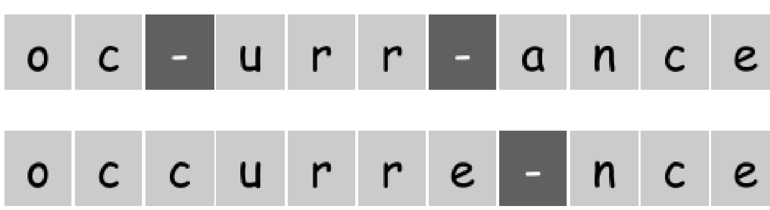
- **ocurrance**
- **occurrence**



5 mismatches, 1 gap



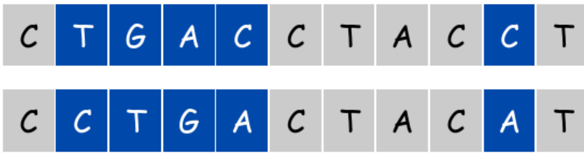
1 mismatch, 1 gap



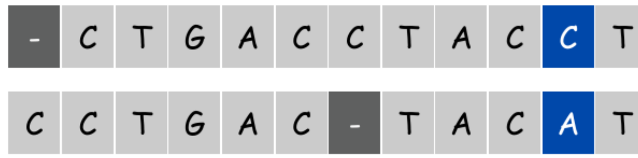
0 mismatches, 3 gaps

Edit distance

[Levenshtein 1966, Needleman-Wunsch 1970, Smith-Waterman 1981]
Gap penalty **δ;** mismatch penalty $\boldsymbol{\alpha_{pq}}$.
Cost = sum of gap and mismatch penalties.



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



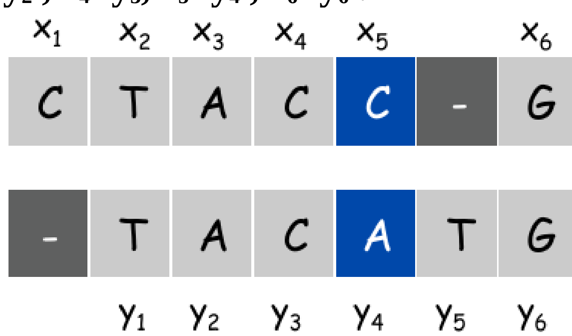
$2\delta + \alpha_{CA}$

Sequence Alignment

Goal: Given two strings $\mathbf{X = x_1\, x_2 \ldots x_m}$ and $\mathbf{Y = y_1\, y_2 \ldots y_n}$ we should find alignment of minimum cost.
Definition. An alignment **M** is a set of ordered pairs $\mathbf{x_i\text{-}y_j}$ such that each item occurs in at most one pair and no crossings.
Definition. The pair $\mathbf{x_i\text{-}y_j}$ and $\mathbf{x_{i'}\text{-}y_{j'}}$ cross if $\mathbf{i < i'}$, but $\mathbf{j > j'}$.

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

Example: **CTACCG** vs. **TACATG**.
Solution: $\mathbf{M = x_2\text{-}y_1,\ x_3\text{-}y_2,\ x_4\text{-}y_3,\ x_5\text{-}y_4,\ x_6\text{-}y_6}$ **.**



Sequence Alignment: Problem Structure

Definition. OPT(i, j) = min cost of aligning strings $\mathbf{x_1\ x_2\ ...\ x_i}$ and $\mathbf{y_1\ y_2\ ...\ y_j}$.

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

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

Sequence Alignment: Algorithm

```
Sequence-Alignment(m, n, x1 x2 ...xm, y1 y2 ...y2, δ, α) {
    for i = 0 to m
        M[0, i] = iδ
    for j = 0 to n
        M[j, 0] = jδ
    for i = 1 to m
        for j = 1 to n
            M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                      δ + M[i-1, j],
                      δ + M[i, j-1])
    return M[m, n]
}
```

Analysis. **O(mn)** time and space.

English words or sentences: **m, n ≤ 10**.

Sequence Alignment: Linear Space

So can we avoid using quadratic space?

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

 Compute OPT(i, •) from OPT(i-1, •).

 No longer a simple way to recover alignment itself.

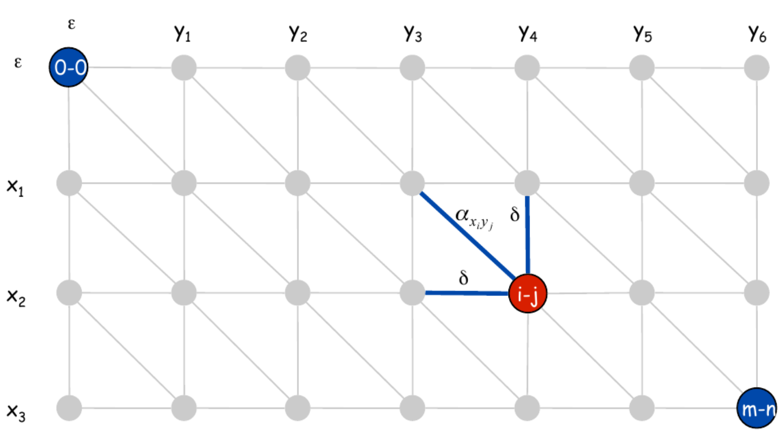
Theorem. [Hirschberg, 1975] Optimal alignment in O(m + n) space and O(mn) time.

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

Sequence Alignment: Linear Space

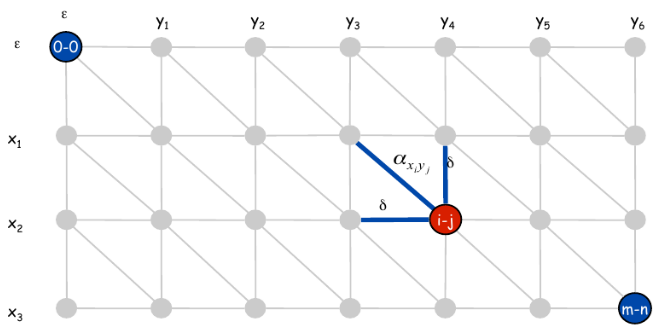
Edit distance graph.

Let f(i, j) be shortest path from (0,0) to (i, j). Observation: f(i, j) = OPT(i, j).

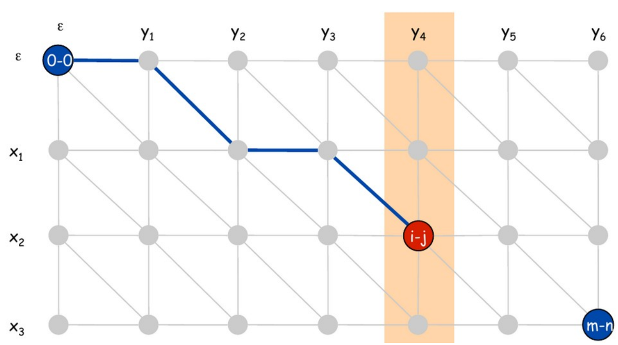


Claim. f(i, j) = OPT(i, j).
Proof. (by induction on i + j)
 Base case: f(0, 0) = OPT(0, 0) = 0.
 Inductive step: assume f(i', j') = OPT(i', j') for all i' + j' < i + j.
 Last edge on path to (i, j) is either from (i-1, j-1), (i-1, j), or (i, j-1).

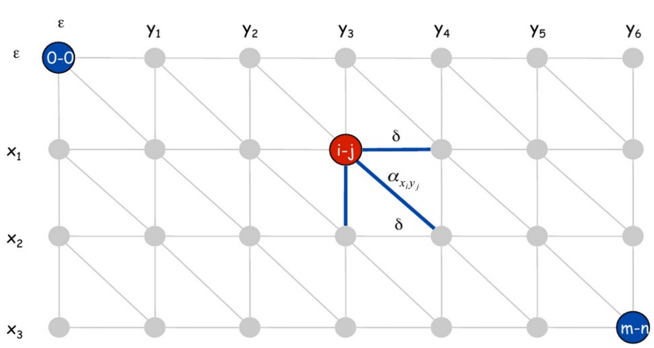
$$
\begin{aligned}
f(i,j) &= \min\{\alpha_{x_i y_j} + f(i-1, j-1), \delta + f(i-1, j), \delta + f(i, j-1)\} \\
&= \min\{\alpha_{x_i y_j} + OPT(i-1, j-1), \delta + OPT(i-1, j), \delta + OPT(i, j-1)\} \\
&= OPT(i,j)
\end{aligned}
$$



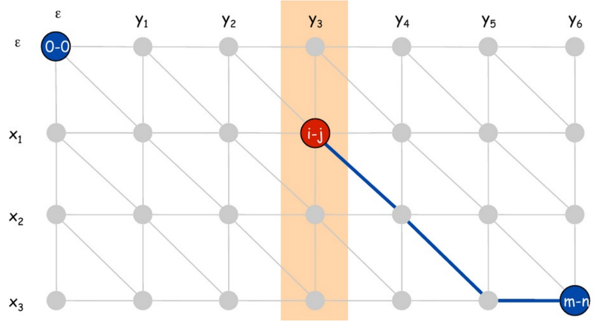
Edit distance graph.
 Let f(i, j) be shortest path from (0,0) to (i, j).
 Can compute f (•, j) for any j in O(mn) time and O(m + n) space.



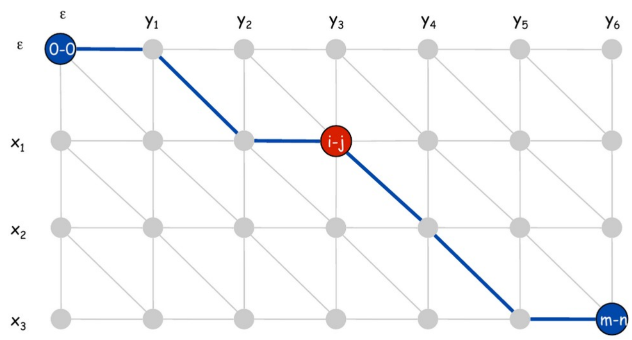
 Let g(i, j) be shortest path from (i, j) to (m, n).
 Can compute by reversing the edge orientations and inverting the roles of (0, 0) and (m, n)



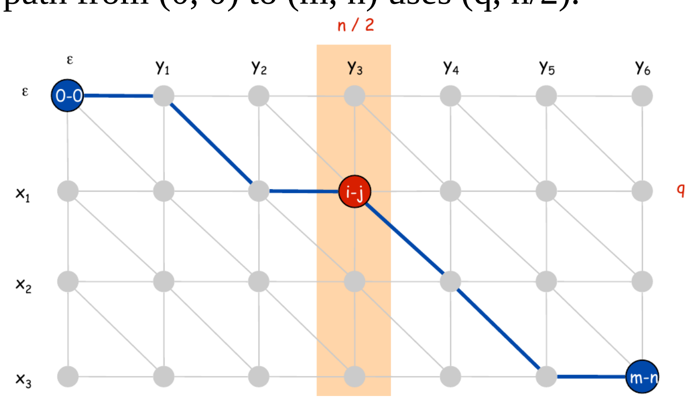
Let g(i, j) be shortest path from (i, j) to (m, n).
Can compute g(•, j) for any j in O(mn) time and O(m + n) space.



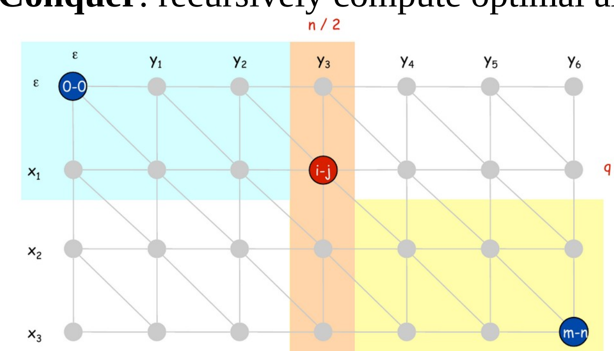
Observation 1. The cost of the shortest path that uses (i, j) is f(i, j) + g(i, j).



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).



Divide: find index q that minimizes f(q, n/2) + g(q, n/2) using DP. Align xq and yn/2.
Conquer: recursively compute optimal alignment in each piece.



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.

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(•, n/2) and g (•, 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:
$T(m, 2) \leq cm$
T(2,T(m,n)n)≤≤cn
$cmn + T(q, n/2) + T(m - q, n/2)$
Base cases: m = 2 or n = 2.
Inductive hypothesis: for m’<m
€ or n’<n, T(m’, n’) ≤ 2cm’n’.

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