CS 401

Dynamic Programming

Sequence Alignment / Shortest Path

Xiaorui Sun

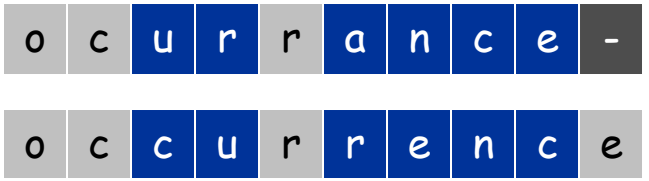
Sequence Alignment

Word Alignment

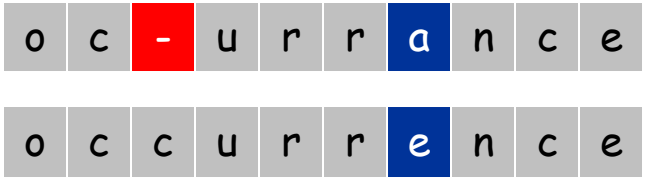
How similar are two strings?

ocurrance

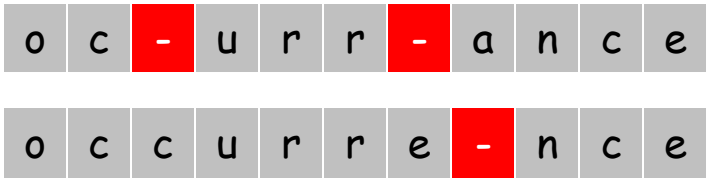
occurrence



6 mismatches, 1 gap



1 mismatch, 1 gap



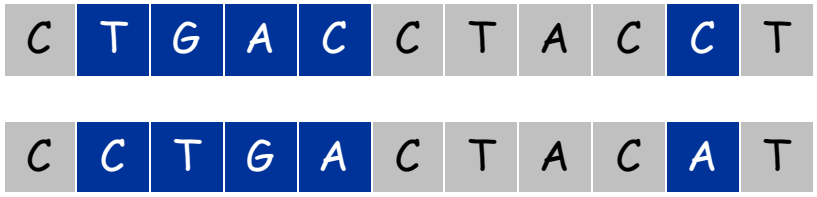
0 mismatches, 3 gaps

Edit Distance

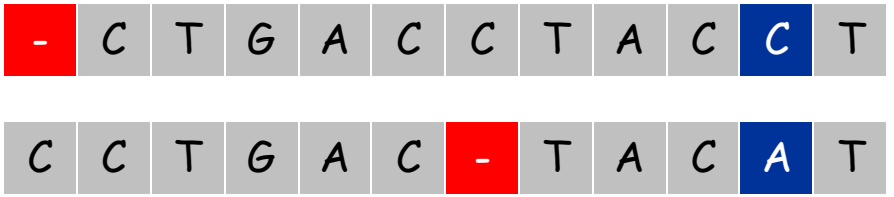
Edit distance. [Levenshtein 1966, Needleman-Wunsch 1970]
Cost = # of gaps + #mismatches.

Applications.

- Basis for Unix diff and Word correct in editors.
- Speech recognition.
- Computational biology.



Cost: 5



Cost: 3

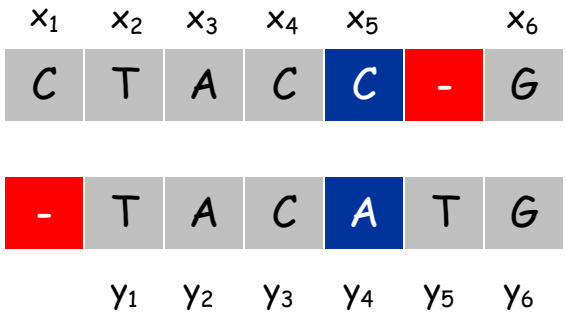
Sequence Alignment

Given two strings $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$ find an alignment with minimum number of mismatch and gaps.

An alignment is a set of ordered pairs $(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2}), \ldots$ such that $i_1 < i_2 < \cdots$ and $j_1 < j_2 < \cdots$

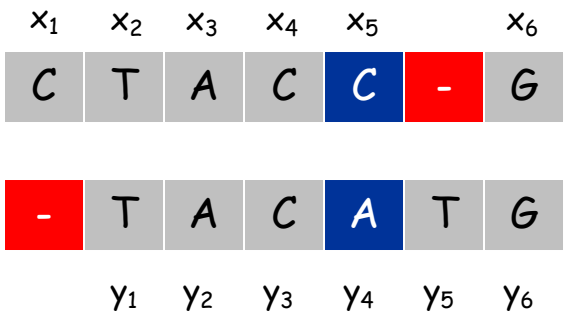
Example: CTACCG vs. TACATG.
Sol: We aligned
x_2-y_1, x_3-y_2, x_4-y_3, x_5-y_4, x_6-y_6.

So, the cost is 3.



Optimal Substructure for String Alignment

Example: CTACCG **vs.** TACATG.



Observation:

- Without last column, it is optimal alignment for CTACC **vs.** TACAT
- Without last two columns, it is optimal alignment for CTACC **vs.** TACA
- …

DP for Sequence Alignment

Let $OPT(i, j)$ be min cost of aligning $x_1, \ldots, x_i$ and $y_1, \ldots, y_j$

Case 1: OPT matches x_i, y_j

- Then, pay mis-match cost if $x_i \neq y_j$ + min cost of aligning $x_1, \ldots, x_{i-1}$ and $y_1, \ldots, y_{j-1}$ i.e., $OPT(i-1, j-1)$

Case 2: OPT leaves x_i unmatched

- Then, pay gap cost for x_i + $OPT(i-1, j)$

Case 3: OPT leaves y_j unmatched

- Then, pay gap cost for y_j + $OPT(i, j-1)$

```
M[i, j] = min( (x_i=y_j ? 0:1) + M[i-1, j-1],
               1 + M[i-1, j],
               1 + M[i, j-1])
```

Induction

What is the order of induction? (i.e. why there is no loop?)

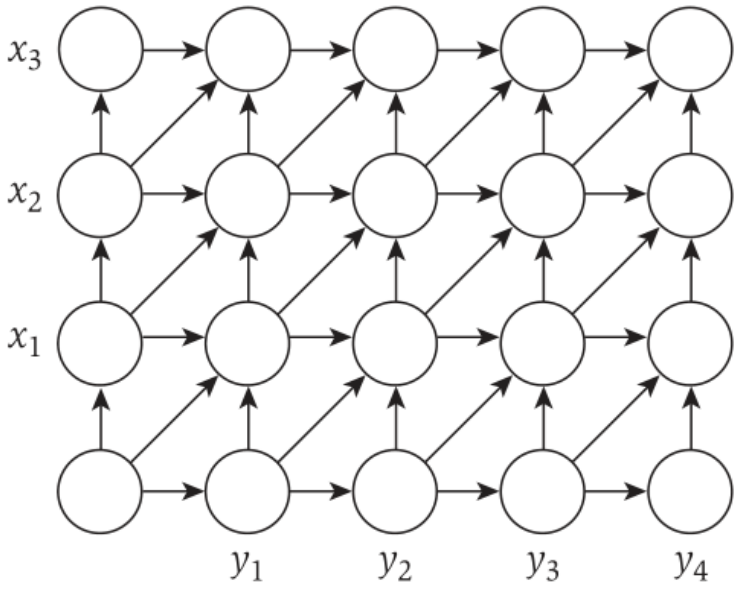


Figure 6.17 A graph-based picture of sequence alignment.

Bottom-up DP

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn) {
  for i = 0 to m
     M[0, i] = i
  for j = 0 to n
     M[j, 0] = j

  for i = 1 to m
     for j = 1 to n
        M[i, j] = min( (xi=yj ? 0:1) + M[i-1, j-1],
                       1 + M[i-1, j],
                       1 + M[i, j-1])
  return M[m, n]
}
```

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

English words or sentences: m, n $\leq$ 10,..,20.

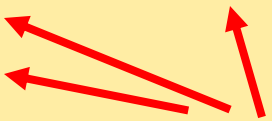
Computational biology: m = n = 100,000. 10 billions ops OK, but 10GB array?

Optimizing Memory

If we are not using strong induction in the DP, we just need to use the last (row) of computed values.

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn) {
   for i = 0 to m
      M[0, i] = i
   for j = 0 to n
      M[j, 0] = j

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min( (xi=yj ? 0:1) + M[i-1, j-1],
                        1 + M[i-1, j],
                        1 + M[i, j-1])
   return M[m, n]
}
```



Just need $i-1, i$ rows to compute M[i,j]

DP with $O(m+n)$ memory

- Keep an Old array containing values of the last row
- Fill out the new values in a New array
- Copy new to old at the end of the loop

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn) {
  for i = 0 to m
     O[i] = i
  for i = 1 to m
     N[0]=i
     for j = 1 to n
        N[j] = min( (xi=yj ? 0:1) + O[j-1],
                    1 + O[j],
                    1 + N[j-1])
     for j = 1 to n
        O[j]=N[j]
  return N[n]
}
```

M[i-1, j-1]

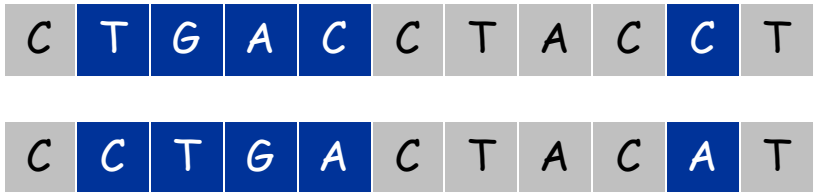
M[i-1, j]

M[i, j-1]

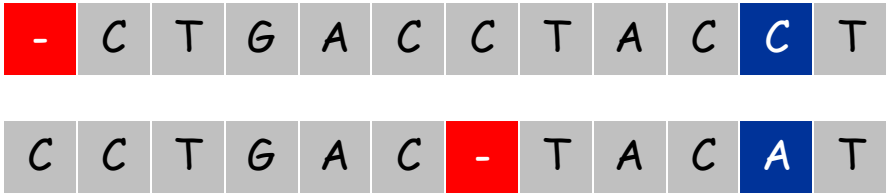
Generalize edit distance

What if the cost of mismatches and gaps are different?

If the cost of mismatch is 1 and the cost of gap is 10,



Cost: 5



Cost: 21

Generalize dynamic programming for edit distance

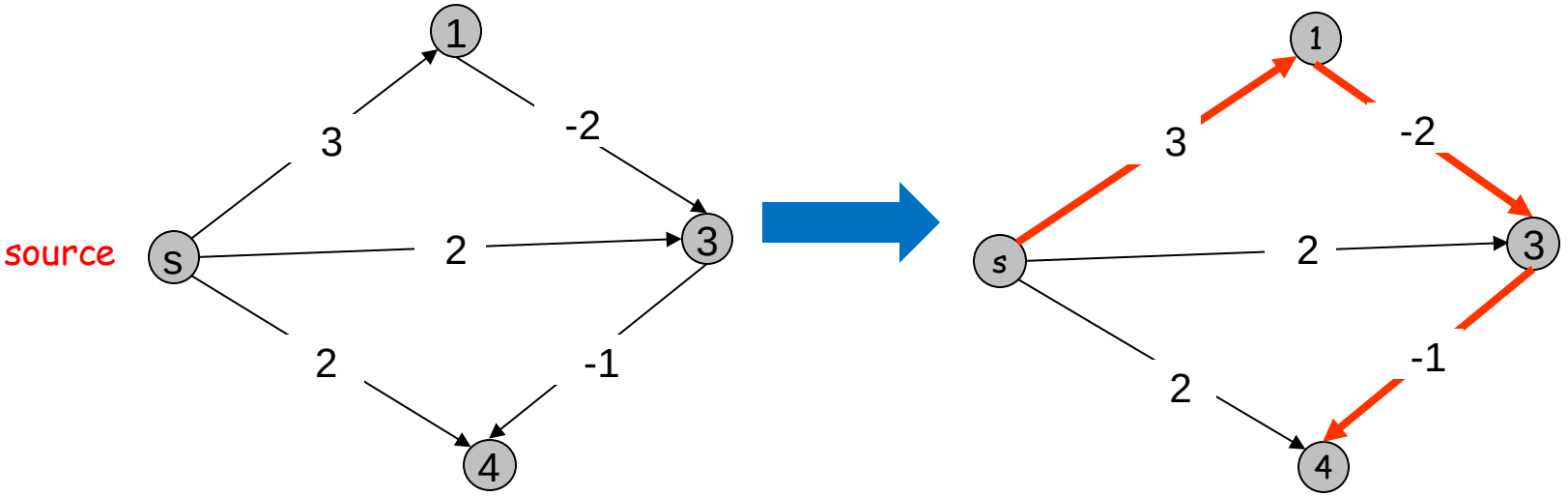
Shortest Paths with Negative Edge Weights

Shortest Paths with Neg Edge Weights

Given a weighted directed graph $G = (V, E)$ and a source vertex s, where the weight of edge (u,v) is $c_{u,v}$ **(that can be negative)**

Goal: Find the shortest path from s to all vertices of G.

Recall that Dikjstra's Algorithm fails when weights are negative

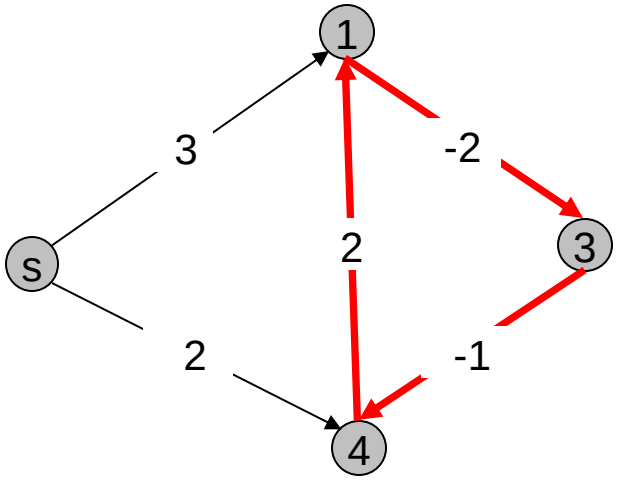


Impossibility on Graphs with Neg Cycles

Observation: No solution exists if G has a negative cycle.

This is because we can minimize the length by going over the cycle again and again.

So, suppose G does not have a negative cycle.



DP for Shortest Path (First Attempt)

Def: Let $OPT(v)$ be the length of the shortest s - v path

$$OPT(v) = \begin{cases} 0 & \text{if } v = s \\ \min_{u:(u,v) \text{ an edge}} OPT(u) + c_{u,v} \end{cases}$$

The formula is correct. But it is not clear how to compute it.

DP for Shortest Path

Def: Let $OPT(v, i)$ be the length of the shortest s - v path with at most i edges.

Let us characterize $OPT(v, i)$.

Case 1: $OPT(v, i)$ path has less than i edges.

- Then, $OPT(v, i) = OPT(v, i - 1)$.

Case 2: $OPT(v, i)$ path has exactly i edges.

- Let $s, v_1, v_2, \dots, v_{i-1}, v$ be the $OPT(v, i)$ path with i edges.
- Then, $s, v_1, \dots, v_{i-1}$ must be the shortest s - v_{i-1} path with at most $i - 1$ edges. So,

$$OPT(v, i) = OPT(v_{i-1}, i - 1) + c_{v_{i-1}, v}$$

DP for Shortest Path

Def: Let $OPT(v,i)$ be the length of the shortest s - v path with at most i edges.

$$OPT(v,i) = \begin{cases} 0 & \text{if } v = s \\ \infty & \text{if } v \neq s, i = 0 \\ \min(OPT(v,i-1), \min_{u:(u,v) \text{ an edge}} OPT(u,i-1) + c_{u,v}) & \end{cases}$$

So, for every v, $OPT(v,?)$ is the shortest path from s to v.

But how long do we have to run?

Since G has no negative cycle, it has at most $n-1$ edges. So, $OPT(v,n-1)$ is the answer.

Bellman Ford Algorithm

```
for v=1 to n
    if v ≠ s then
        M[v,0]=∞
M[s,0]=0.

for i=1 to n-1
    for v=1 to n
        M[v,i]=M[v,i-1
        for every edge
            M[v,i]=min(
```

	Complexity	Author
	$O(n^4)$	Shimbel (1955) [30]
	$O(Wn^2m)$	Ford (1956) [14]
*	$O(nm)$	Bellman (1958) [1], Moore (1959) [25]
	$O(n^{\frac{3}{4}}m\log W)$	Gabow (1983) [9]
	$O(\sqrt{n}m\log(nW))$	Gabow and Tarjan (1989) [10]
*	$O(\sqrt{n}m\log(W))$	Goldberg (1993) [12]
*	$\tilde{O}(Wn^{\omega})$	Sankowski (2005) [27] Yuster and Zwick (2005) [35]
*	$\tilde{O}(m^{10/7}\log W)$	Cohen, Madry, Sankowski, Vladu (2016)

Table 1: The complexity results for the SSSP problem with negative weights (* indicates asymptotically the best bound for some range of parameters).

$m^{1+o(1)}\log W$ algorithm
By Bernstein, Nanongkai, and Wulff-Nilsen; Chen, Kyng, Liu, Peng, Gutenberg, Sachdeva 2022

Running Time: $O(nm)$

Can we test if G has negative cycles?

Yes, run for i=1…3n and see if the M[v,n-1] is different from M[v,3n]

DP Techniques Summary

Principle:

- Optimal substructure: Remove certain part of the optimal solution (for the entire problem) is an optimal solution of a subproblem
- Carefully define a collection of subproblems. Typically, only a polynomial number of subproblems
- Parameterization/Memorization

Recipe:

- Find optimal substructure by investigating the optimal solution
- Find out additional assumptions/variables/subproblems that you need to do the induction
- Strengthen the hypothesis and define w.r.t. new subproblems

Dynamic programming techniques.

- Adding a new variable: knapsack.
- Order subproblems in the right way: RNA secondary structure/String Alignment