

## CS 580: Algorithm Design and Analysis

Jeremiah Blocki  
Purdue University  
Spring 2019

Announcement: Homework 3 due February 15<sup>th</sup> at 11:59PM  
Midterm Exam 1: Wed, Feb 20 (8PM-10PM) @ EE 170  
Final Exam: Mon 04/29 (08:00am - 10:00am) @ PHYS 223

### Recap: Dynamic Programming

**Key Idea:** Express optimal solution in terms of solutions to smaller sub problems

**Example 1: Knapsack Problem**

- Two Dimensional Solution  $OPT(j, w)$
- $OPT(j, w) = \max\{v_j + OPT(j-1, w-w_j), OPT(j-1, w)\}$
- Case 1: Optimal solution includes item  $j$  with value  $v_j$ 
  - Add item  $j$  and reduce remaining capacity to  $w-w_j$
- Case 2: Optimal solution does not include item  $j$

**Example 2: RNA Secondary Structure**

- Goal: Maximize number of matched base pairs
- Constraints: No Sharp Turns, Watson-Crick Complements, No Crossing Edges
- $OPT(i, j)$  = maximum number of base pairs in a secondary structure of the substring  $b_i b_{i+1} \dots b_j$
- $OPT(i, j) = \max\{OPT(i, j-1), \max_t \{1 + OPT(i, t-1) + OPT(t+1, j-1)\}\}$
- $b_i$  unpaired                       $b_t$  paired with  $b_j$

## 6.6 Sequence Alignment

### String Similarity

How similar are two strings?

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

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

- Gap penalty  $\delta$ ; mismatch penalty  $\alpha_{pq}$ .
- Cost = sum of gap and mismatch penalties.

c t g a c c t a c c t    - c t g a c c t a c c t  
c c t g a c t a c a t    c c t g a c - t a c a t

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

**Applications.**

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

### Sequence Alignment

**Goal:** Given two strings  $X = x_1 x_2 \dots x_m$  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  $j > j'$ .

$$\text{cost}(M) = \sum_{(x_i, y_j) \in M} \alpha_{x_i y_j} + \sum_{j: x_j \text{ unmatched}} \delta + \sum_{j: y_j \text{ unmatched}} \delta$$

**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  
-   t   a   c   a   t   g  
 $y_1$   $y_2$   $y_3$   $y_4$   $y_5$   $y_6$

### Sequence Alignment: Problem Structure

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

- Case 1: OPT 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: OPT 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: OPT leaves  $y_j$  unmatched.
  - pay gap for  $y_j$  and min cost of aligning  $x_1 x_2 \dots x_i$  and  $y_1 y_2 \dots y_{j-1}$ .

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

7

### Sequence Alignment: Algorithm

```

Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α) {
  for i = 0 to m
    M[i, 0] = iδ
  for j = 0 to n
    M[0, j] = 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.**  $\Theta(mn)$  time and space.  
**English words or sentences:**  $m, n \leq 10$ .  
**Computational biology:**  $m = n = 100,000$ .  
 10 billions ops OK, but 10GB array?

8

## 6.7 Sequence Alignment in Linear Space

### Sequence Alignment: Linear Space

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

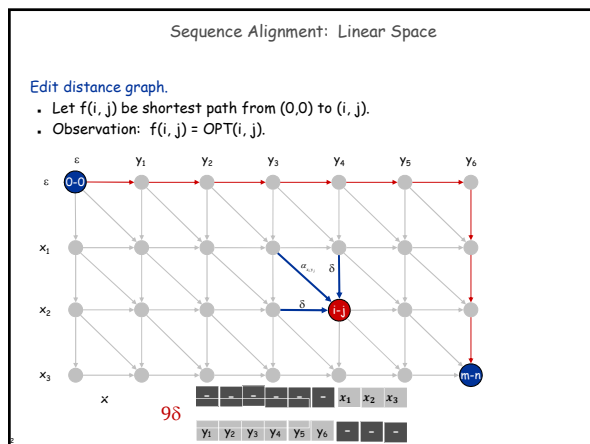
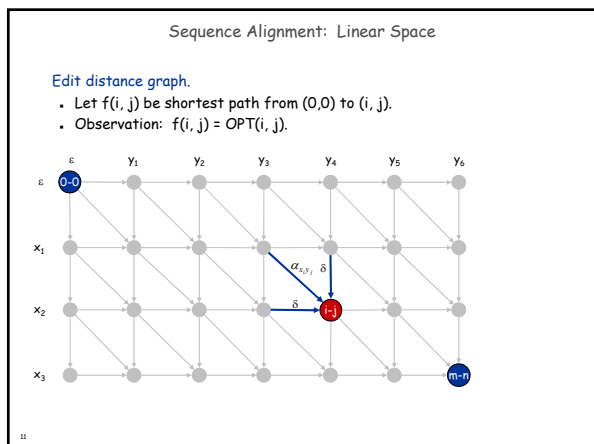
**Easy.** Optimal value in  $O(m + n)$  space and  $O(mn)$  time.

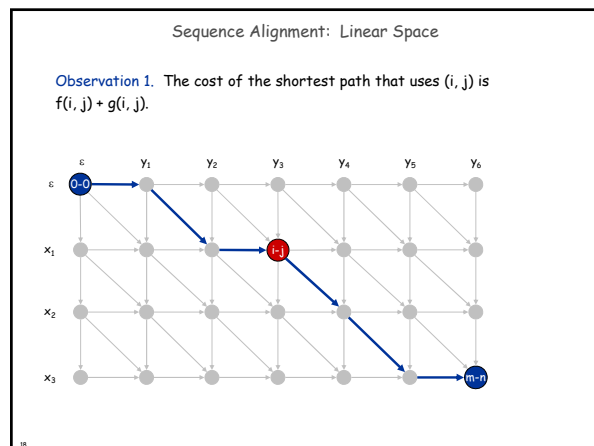
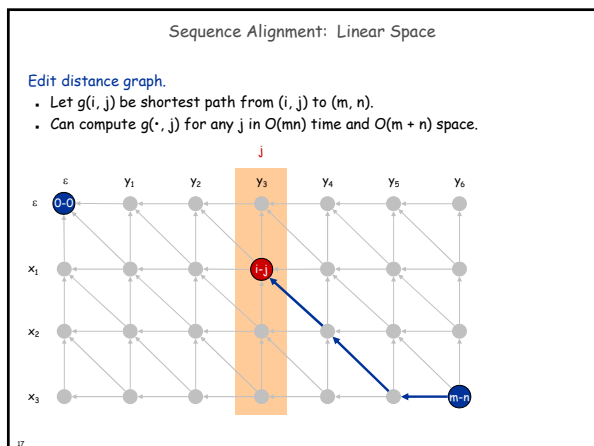
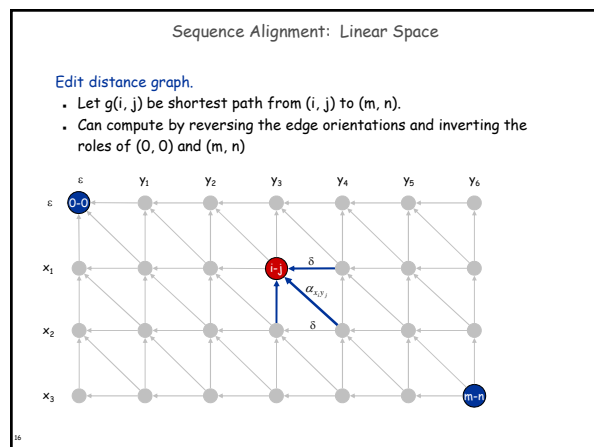
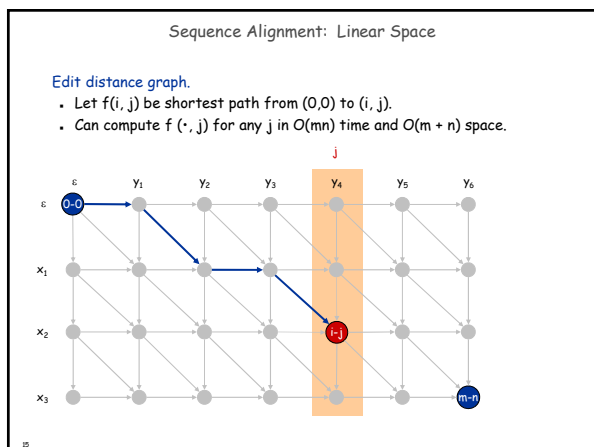
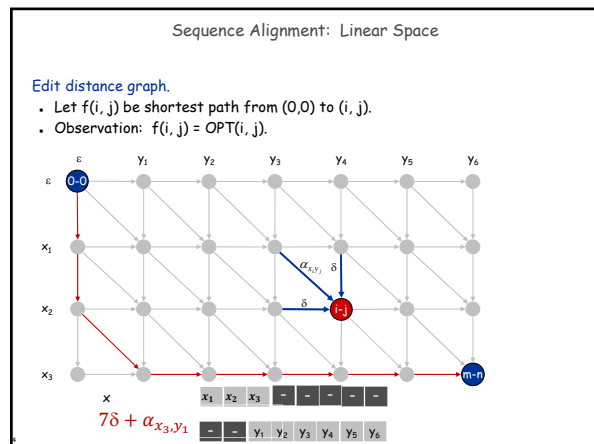
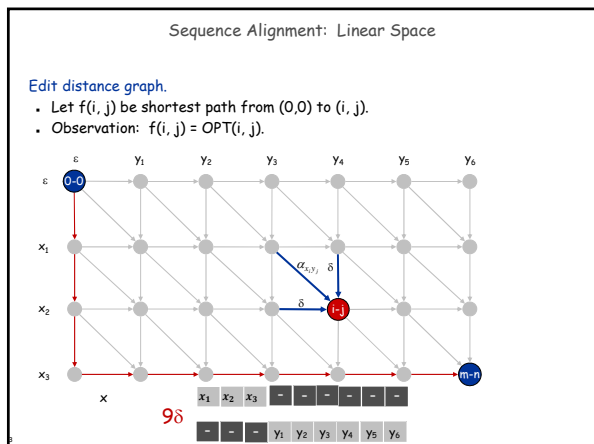
- Compute  $OPT(i, \cdot)$  from  $OPT(i-1, \cdot)$ .
- No longer a simple way to recover alignment itself.

**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.

10





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  $(q, n/2)$ .

19

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.

20

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.

21

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.
- 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:  $T(m, n) \leq 2cmn$ .

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

22

## 6.8 Shortest Paths

Shortest Paths

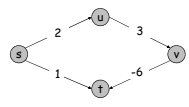
**Shortest path problem.** Given a directed graph  $G = (V, E)$ , with edge weights  $c_{vw}$ , find shortest path from node  $s$  to node  $t$ .  
 ↙ allow negative weights

**Ex.** Nodes represent agents in a financial setting and  $c_{vw}$  is cost of transaction in which we buy from agent  $v$  and sell immediately to  $w$ .

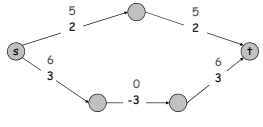
24

### Shortest Paths: Failed Attempts

**Dijkstra.** Can fail if negative edge costs.



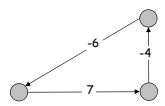
**Re-weighting.** Adding a constant to every edge weight can fail.



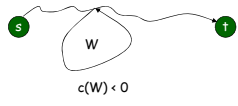
25

### Shortest Paths: Negative Cost Cycles

**Negative cost cycle.**



**Observation.** If some path from s to t contains a negative cost cycle, there does not exist a shortest s-t path; otherwise, there exists one that is simple.



26

### Shortest Paths: Dynamic Programming

**Def.**  $OPT(i, v)$  = length of shortest v-t path P using at most i edges.

- Case 1: P uses at most i-1 edges.
  - $OPT(i, v) = OPT(i-1, v)$
- Case 2: P uses exactly i edges.
  - if (v, w) is first edge, then OPT uses (v, w), and then selects best w-t path using at most i-1 edges

$$OPT(i, v) = \begin{cases} 0 & \text{if } i = 0 \\ \min \left\{ OPT(i-1, v), \min_{(v,w) \in E} \{ OPT(i-1, w) + c_{vw} \} \right\} & \text{otherwise} \end{cases}$$

**Remark.** By previous observation, if no negative cycles, then  $OPT(n-1, v)$  = length of shortest v-t path.

27

### Shortest Paths: Implementation

```

Shortest-Path(G, t) {
  foreach node v ∈ V
    M[0, v] ← ∞
  M[0, t] ← 0

  for i = 1 to n-1
    foreach node v ∈ V
      M[i, v] ← M[i-1, v]
      foreach edge (v, w) ∈ E
        M[i, v] ← min { M[i, v], M[i-1, w] + c_w }
}
    
```

**Analysis.**  $\Theta(mn)$  time,  $\Theta(n^2)$  space.

**Finding the shortest paths.** Maintain a "successor" for each table entry.

28

### Shortest Paths: Practical Improvements

**Practical improvements.**

- Maintain only one array  $M[v]$  = shortest v-t path that we have found so far.
- No need to check edges of the form (v, w) unless  $M[w]$  changed in previous iteration.

**Theorem.** Throughout the algorithm,  $M[v]$  is length of some v-t path, and after i rounds of updates, the value  $M[v]$  is no larger than the length of shortest v-t path using  $\leq i$  edges.

**Overall impact.**

- Memory:  $O(m + n)$ .
- Running time:  $O(mn)$  worst case, but substantially faster in practice.

29

### Bellman-Ford: Efficient Implementation

```

Push-Based-Shortest-Path(G, s, t) {
  foreach node v ∈ V {
    M[v] ← ∞
    successor[v] ← ∅
  }

  M[t] = 0
  for i = 1 to n-1 {
    foreach node w ∈ V {
      if (M[w] has been updated in previous iteration) {
        foreach node v such that (v, w) ∈ E {
          if (M[v] > M[w] + c_w) {
            M[v] ← M[w] + c_w
            successor[v] ← w
          }
        }
      }
    }
    If no M[w] value changed in iteration i, stop.
  }
}
    
```

30

## 6.9 Distance Vector Protocol

### Distance Vector Protocol

**Communication network.**

- Node  $\approx$  router.
- Edge = direct communication link.
- Cost of edge  $\approx$  delay on link. ← naturally nonnegative, but Bellman-Ford used anyway!

**Dijkstra's algorithm.** Requires global information of network.

**Bellman-Ford.** Uses only local knowledge of neighboring nodes.

**Synchronization.** We don't expect routers to run in lockstep. The order in which each `foreach` loop executes is not important. Moreover, algorithm still converges even if updates are asynchronous.

32

### Distance Vector Protocol

**Distance vector protocol.**

- Each router maintains a vector of shortest path lengths to every other node (distances) and the first hop on each path (directions).
- Algorithm: each router performs  $n$  separate computations, one for each potential destination node.
- "Routing by rumor."

**Ex.** RIP, Xerox XNS RIP, Novell's IPX RIP, Cisco's IGRP, DEC's DNA Phase IV, AppleTalk's RTMP.

**Caveat.** Edge costs may **change** during algorithm (or fail completely).

33

### Path Vector Protocols

**Link state routing.**

- Each router also stores the entire path. not just the distance and first hop
- Based on Dijkstra's algorithm.
- Avoids "counting-to-infinity" problem and related difficulties.
- Requires significantly more storage.

**Ex.** Border Gateway Protocol (BGP), Open Shortest Path First (OSPF).

34

## 6.10 Negative Cycles in a Graph

### Detecting Negative Cycles

**Lemma.** If  $OPT(n,v) = OPT(n-1,v)$  for all  $v$ , then no negative cycles.  
**Pf.** Bellman-Ford algorithm.

**Lemma.** If  $OPT(n,v) < OPT(n-1,v)$  for some node  $v$ , then (any) shortest path from  $v$  to  $t$  contains a cycle  $W$ . Moreover  $W$  has negative cost.

**Pf.** (by contradiction)

- Since  $OPT(n,v) < OPT(n-1,v)$ , we know  $P$  has exactly  $n$  edges.
- By pigeonhole principle,  $P$  must contain a directed cycle  $W$ .
- Deleting  $W$  yields a  $v$ - $t$  path with  $< n$  edges  $\Rightarrow W$  has negative cost.

35

### Detecting Negative Cycles

**Theorem.** Can detect negative cost cycle in  $O(mn)$  time.

- Add new node  $t$  and connect all nodes to  $t$  with 0-cost edge.
- Check if  $OPT(n, v) = OPT(n-1, v)$  for all nodes  $v$ .
  - if yes, then no negative cycles
  - if no, then extract cycle from shortest path from  $v$  to  $t$

37

### Detecting Negative Cycles: Application

**Currency conversion.** Given  $n$  currencies and exchange rates between pairs of currencies, is there an arbitrage opportunity?

**Remark.** Fastest algorithm very valuable!

38

### Detecting Negative Cycles: Summary

**Bellman-Ford.**  $O(mn)$  time,  $O(m + n)$  space.

- Run Bellman-Ford for  $n$  iterations (instead of  $n-1$ ).
- Upon termination, Bellman-Ford successor variables trace a negative cycle if one exists.
- See p. 304 for improved version and early termination rule.

39