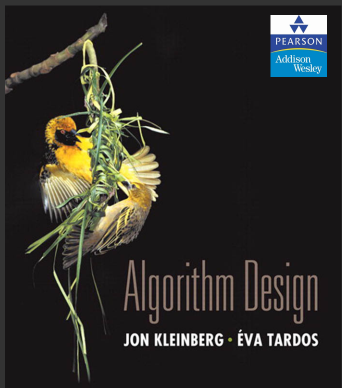


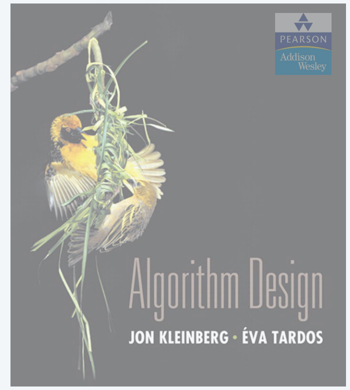
6. Dynamic Programming II

- *sequence alignment*
- *Hirschberg's algorithm*
- *Bellman–Ford–Moore algorithm*
- *distance-vector protocols*
- *negative cycles*

Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on 5/19/21 10:15 AM



Section 6.6

6. Dynamic Programming II

- *sequence alignment*
- *Hirschberg's algorithm*
- *Bellman–Ford–Moore algorithm*
- *distance-vector protocols*
- *negative cycles*

String similarity

Q. How similar are two strings?

Ex. ocurrance and occurrence.

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

C	T	–	G	A	C	C	T	A	C	G
C	T	G	G	A	C	G	A	A	C	G

cost = $\delta + \alpha_{CG} + \alpha_{TA}$ ← assuming $\alpha_{AA} = \alpha_{CC} = \alpha_{GG} = \alpha_{TT} = 0$

Applications. Bioinformatics, spell correction, machine translation, speech recognition, information extraction, ...

```
Spokesperson confirms     senior government adviser was found
Spokesperson said     the senior                adviser was found
```

BLOSUM matrix for proteins

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	7	-3	-3	-3	-1	-2	-2	0	-3	-3	-3	-1	-2	-4	-1	2	0	-5	-4	-1
R	-3	9	-1	-3	-6	1	-1	-4	0	-5	-4	3	-3	-5	-3	-2	-2	-5	-4	-4
N	-3	-1	9	2	-5	0	-1	-1	1	-6	-6	0	-4	-6	-4	1	0	-7	-4	-5
D	-3	-3	2	10	-7	-1	2	-3	-2	-7	-7	-2	-6	-6	-3	-1	-2	-8	-6	-6
C	-1	-6	-5	-7	13	-5	-7	-6	-7	-2	-3	-6	-3	-4	-6	-2	-2	-5	-5	-2
Q	-2	1	0	-1	-5	9	3	-4	1	-5	-4	2	-1	-5	-3	-1	-1	-4	-3	-4
E	-2	-1	-1	2	-7	3	8	-4	0	-6	-6	1	-4	-6	-2	-1	-2	-6	-5	-4
G	0	-4	-1	-3	-6	-4	-4	9	-4	-7	-7	-3	-5	-6	-5	-1	-3	-6	-6	-6
H	-3	0	1	-2	-7	1	0	-4	12	-6	-5	-1	-4	-2	-4	-2	-3	-4	3	-5
I	-3	-5	-6	-7	-2	-5	-6	-7	-6	7	2	-5	2	-1	-5	-4	-2	-5	-3	4
L	-3	-4	-6	-7	-3	-4	-6	-7	-5	2	6	-4	3	0	-5	-4	-3	-4	-2	1
K	-1	3	0	-2	-6	2	1	-3	-1	-5	-4	8	-3	-5	-2	-1	-1	-6	-4	-4
M	-2	-3	-4	-6	-3	-1	-4	-5	-4	2	3	-3	9	0	-4	-3	-1	-3	-3	1
F	-4	-5	-6	-6	-4	-5	-6	-6	-2	-1	0	-5	0	10	-6	-4	-4	0	4	-2
P	-1	-3	-4	-3	-6	-3	-2	-5	-4	-5	-5	-2	-4	-6	12	-2	-3	-7	-6	-4
S	2	-2	1	-1	-2	-1	-1	-1	-2	-4	-4	-1	-3	-4	-2	7	2	-6	-3	-3
T	0	-2	0	-2	-2	-1	-2	-3	-3	-2	-3	-1	-1	-4	-3	2	8	-5	-3	0
W	-5	-5	-7	-8	-5	-4	-6	-6	-4	-5	-4	-6	-3	0	-7	-6	-5	16	3	-5
Y	-4	-4	-4	-6	-5	-3	-5	-6	3	-3	-2	-4	-3	4	-6	-3	-3	3	11	-3
V	-1	-4	-5	-6	-2	-4	-4	-6	-5	4	1	-4	1	-2	-4	-3	0	-5	-3	7

Dynamic programming: quiz 1

What is edit distance between these two strings?

P A L E T T E   P A L A T E

Assume gap penalty = 2 and mismatch penalty = 1.

A. 1

B. 2

C. 3

D. 4

E. 5

Sequence alignment

Goal. Given two strings $x_1\, x_2 \ldots x_m$ and $y_1\, y_2 \ldots y_n$, find a min-cost alignment.

Def. An alignment M is a set of ordered pairs $x_i - y_j$ such that each character appears in at most one pair and no crossings.

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

Def. The cost of an alignment M is:

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

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

an alignment of CTACCG and TACATG

$M = \{ x_2–y_1, x_3–y_2, x_4–y_3, x_5–y_4, x_6–y_6 \}$

Sequence alignment: problem structure

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

Goal. $OPT(m, n)$.

Case 1. $OPT(i, j)$ matches $x_i – y_j$.

Pay mismatch for $x_i – y_j$ + min cost of aligning $x_1\, x_2 \ldots x_{i-1}$ and $y_1\, y_2 \ldots y_{j-1}$.

Case 2a. $OPT(i, j)$ leaves x_i unmatched.

Pay gap for x_i + min cost of aligning $x_1\, x_2 \ldots x_{i-1}$ and $y_1\, y_2 \ldots y_j$.

optimal substructure property (proof via exchange argument)

Case 2b. $OPT(i, j)$ leaves y_j unmatched.

Pay gap for y_j + min cost of aligning $x_1\, x_2 \ldots x_i$ and $y_1\, y_2 \ldots y_{j-1}$.

Bellman equation.

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

Sequence alignment: bottom-up algorithm

SEQUENCE-ALIGNMENT$(m, n, x_1, \ldots, x_m, y_1, \ldots, y_n, \delta, \alpha)$

FOR $i = 0$ TO m
  $M[i, 0] \leftarrow i\,\delta$.
FOR $j = 0$ TO n
  $M[0, j] \leftarrow j\,\delta$.

FOR $i = 1$ TO m
  FOR $j = 1$ TO n
    $M[i, j] \leftarrow \min \{ \alpha_{x_i y_j} + M[i-1, j-1],$
      $\delta + M[i-1, j],$
      $\delta + M[i, j-1] \}$.

already computed

RETURN $M[m, n]$.

Sequence alignment: traceback

		P	A	L	A	T	E
	0	2	4	6	8	10	12
P	2	0	2	4	6	8	10
A	4	2	0	2	4	6	8
L	6	4	2	0	2	4	6
E	8	6	4	2	1	3	4
T	10	8	6	4	3	1	3
T	12	10	8	6	5	3	2
E	14	12	10	8	7	5	3

P	A	L	E	T	T	E
P	A	L	–	A	T	E

1 mismatch, 1 gap

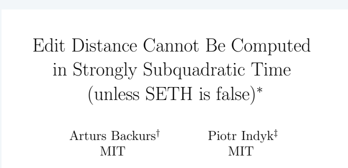
Sequence alignment: analysis

Theorem. The DP algorithm computes the edit distance (and an optimal alignment) of two strings of lengths m and n in $\Theta(mn)$ time and space.
Pf.

- Algorithm computes edit distance.
- Can trace back to extract optimal alignment itself. ▪

Theorem. [Backurs–Indyk 2015] If can compute edit distance of two strings of length n in $O(n^{2-\varepsilon})$ time for some constant $\varepsilon > 0$, then can solve SAT with n variables and m clauses in $poly(m)\ 2^{(1-\delta)\,n}$ time for some constant $\delta > 0$.

which would disprove SETH (strong exponential time hypothesis)


Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false)*

Arturs Backurs† MIT
Piotr Indyk‡ MIT

Dynamic programming: quiz 3

It is easy to modify the DP algorithm for edit distance to…

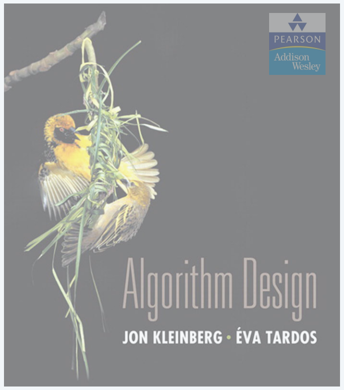
A. Compute edit distance in $O(mn)$ time and $O(m + n)$ space.

B. Compute an optimal alignment in $O(mn)$ time and $O(m + n)$ space.

C. Both A and B.

D. Neither A nor B.

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



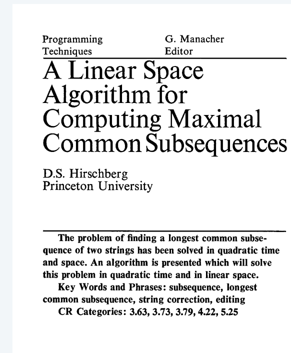
SECTION 6.7

6. DYNAMIC PROGRAMMING II

Sequence alignment in linear space

Theorem. [Hirschberg] There exists an algorithm to find an optimal alignment in $O(mn)$ time and $O(m + n)$ space.

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



Programming Techniques — G. Manacher, Editor

A Linear Space Algorithm for Computing Maximal Common Subsequences

D.S. Hirschberg
Princeton University

The problem of finding a longest common subsequence of two strings has been solved in quadratic time and space. An algorithm is presented which will solve this problem in quadratic time and in linear space.

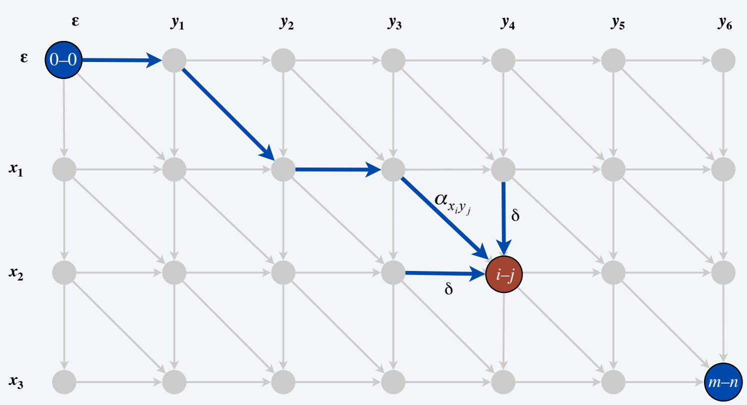
Key Words and Phrases: subsequence, longest common subsequence, string correction, editing

CR Categories: 3.63, 3.73, 3.79, 4.22, 5.25

Hirschberg's algorithm

Edit distance graph.

- Let $f(i, j)$ denote length of shortest path from $(0,0)$ to (i, j).
- Lemma: $f(i, j) = OPT(i, j)$ for all i and j.



Hirschberg's algorithm

Edit distance graph.

- Let $f(i, j)$ denote length of shortest path from $(0,0)$ to (i, j).
- Lemma: $f(i, j) = OPT(i, j)$ for all i and j.

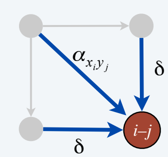
Pf of Lemma. [by strong induction on $i + j$]

- Base case: $f(0, 0) = OPT(0, 0) = 0$.
- Inductive hypothesis: assume true for all (i', j') with $i' + j' < i + j$.
- Last edge on shortest path to (i, j) is from $(i - 1, j - 1)$, $(i - 1, j)$, or $(i, j - 1)$.
- Thus,

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

inductive hypothesis

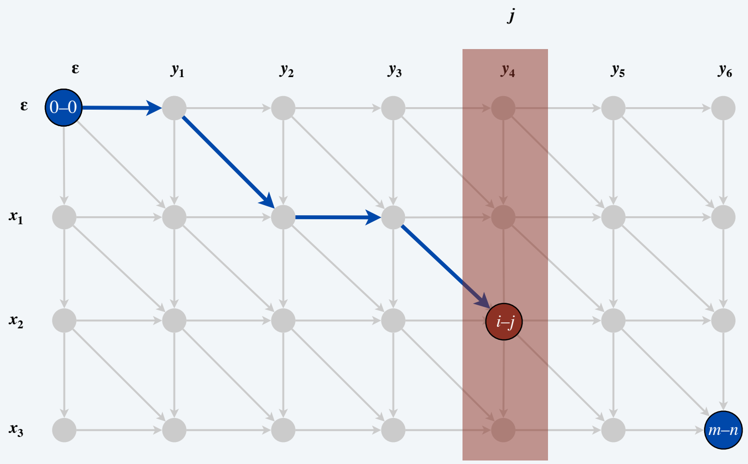
Bellman equation



Hirschberg's algorithm

Edit distance graph.

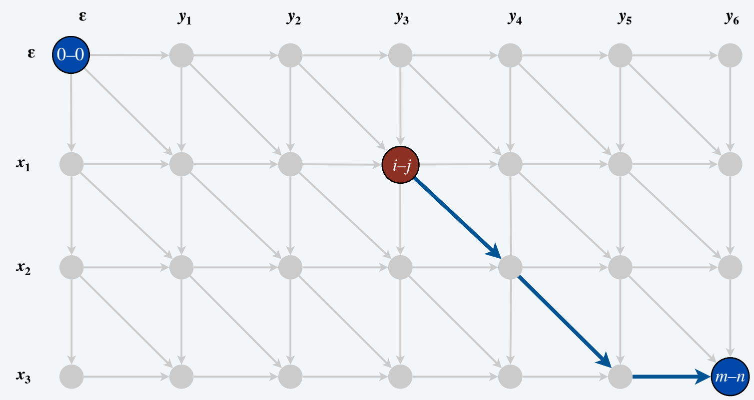
- Let $f(i, j)$ denote length of shortest path from $(0,0)$ to (i, j).
- Lemma: $f(i, j) = OPT(i, j)$ for all i and j.
- Can compute $f(\cdot, j)$ for any j in $O(mn)$ time and $O(m + n)$ space.



Hirschberg's algorithm

Edit distance graph.

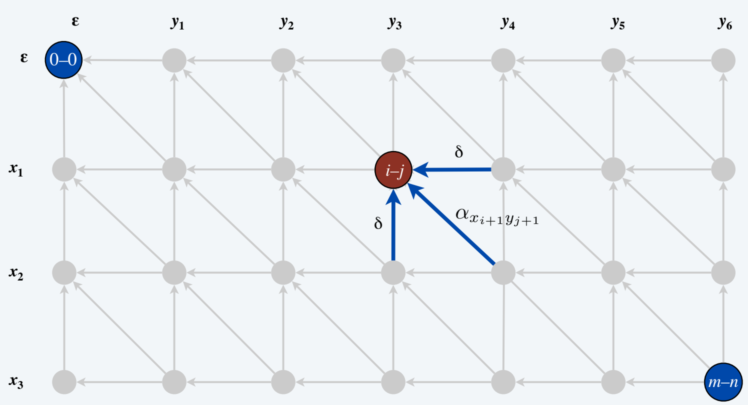
- Let $g(i, j)$ denote length of shortest path from (i, j) to (m, n).



Hirschberg's algorithm

Edit distance graph.

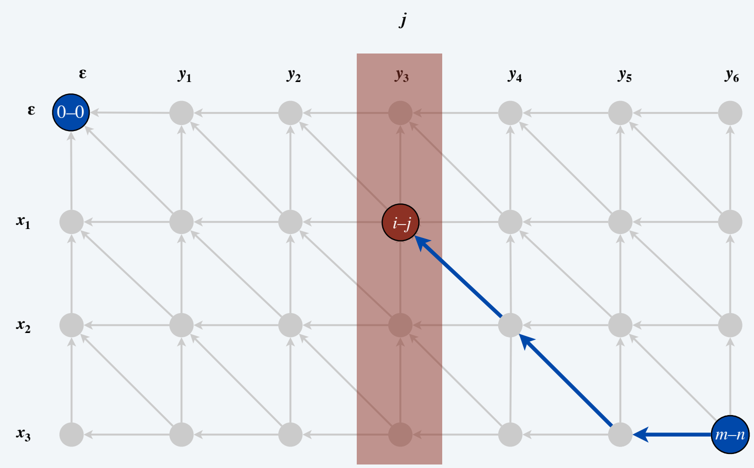
- Let $g(i, j)$ denote length of shortest path from (i, j) to (m, n).
- Can compute $g(i, j)$ by reversing the edge orientations and inverting the roles of $(0, 0)$ and (m, n).



Hirschberg's algorithm

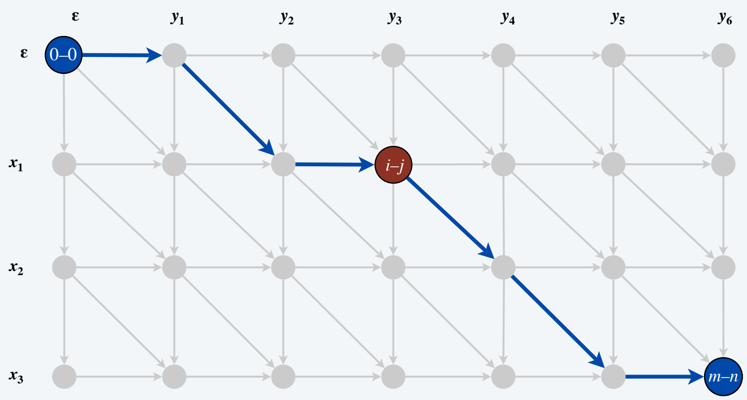
Edit distance graph.

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



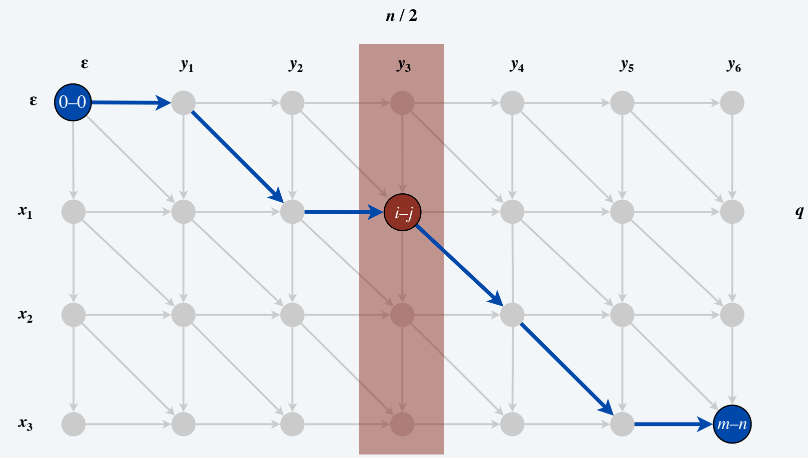
Hirschberg's algorithm

Observation 1. The length of a shortest path that uses (i, j) is $f(i, j) + g(i, j)$.



Hirschberg's algorithm

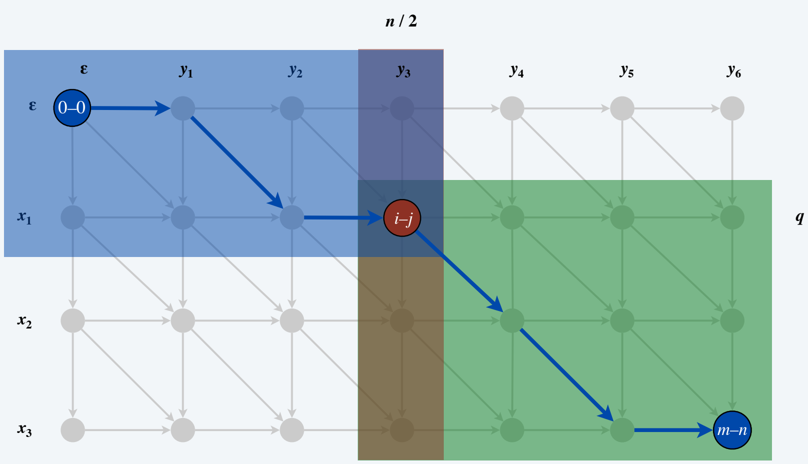
Observation 2. let q be an index that minimizes $f(q, n/2) + g(q, n/2)$. Then, there exists a shortest path from $(0, 0)$ to (m, n) that uses $(q, n/2)$.



Hirschberg's algorithm

Divide. Find index q that minimizes $f(q, n/2) + g(q, n/2)$; save node i–j as part of solution.

Conquer. Recursively compute optimal alignment in each piece.



Hirschberg's algorithm: space analysis

Theorem. Hirschberg's algorithm uses $\Theta(m + n)$ space.

Pf.

- Each recursive call uses $\Theta(m)$ space to compute $f(\cdot, n/2)$ and $g(\cdot, n/2)$.
- Only $\Theta(1)$ space needs to be maintained per recursive call.
- Number of recursive calls $\leq n$. ▪

Dynamic programming: quiz 4

What is the worst-case running time of Hirschberg's algorithm?

A. $O(mn)$

B. $O(mn \log m)$

C. $O(mn \log n)$

D. $O(mn \log m \log n)$

Hirschberg's algorithm: running time analysis warmup

Theorem. Let $T(m, n)$ = max running time of Hirschberg's algorithm on strings of lengths at most m and n. Then, $T(m, n) = O(m\, n \log n)$.

Pf.

- $T(m, n)$ is monotone nondecreasing in both m and n.
- $T(m, n) \le 2\, T(m, n/2) + O(m\, n)$

  $\Rightarrow T(m, n) = O(m\, n \log n)$.

Remark. Analysis is not tight because two subproblems are of size $(q, n/2)$ and $(m-q, n/2)$. Next, we prove $T(m, n) = O(m\, n)$.

Hirschberg's algorithm: running time analysis

Theorem. Let $T(m, n)$ = max running time of Hirschberg's algorithm on strings of lengths at most m and n. Then, $T(m, n) = O(m\, n)$.

Pf. [by strong induction on $m + n$]

- $O(m\, n)$ 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: $T(m, 2) \le c\, m$

  $T(2, n) \le c\, n$

  $T(m, n) \le c\, m\, n + T(q, n/2) + T(m-q, n/2)$
- Claim. $T(m, n) \le 2\, c\, m\, n$.
- Base cases: $m = 2$ and $n = 2$.
- Inductive hypothesis: $T(m, n) \le 2\, c\, m\, n$ for all (m', n') with $m' + n' < m + n$.

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

inductive hypothesis

LONGEST COMMON SUBSEQUENCE

Problem. Given two strings $x_1\, x_2 \ldots x_m$ and $y_1\, y_2 \ldots y_n$, find a common subsequence that is as long as possible.

Alternative viewpoint. Delete some characters from x; delete some character from y; a common subsequence if it results in the same string.

Ex. LCS(GGCACCACG, ACGGCGGATACG) = GGCAACG.

Applications. Unix diff, git, bioinformatics.

Longest common subsequence

Solution 1. Dynamic programming.

Def. $OPT(i, j)$ = length of LCS of prefix strings $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j$.

Goal. $OPT(m, n)$.

Case 1. $x_i = y_j$.

- 1 + length of LCS of $x_1 x_2 \ldots x_{i-1}$ and $y_1 y_2 \ldots y_{j-1}$.

optimal substructure property (proof via exchange argument)

Case 2. $x_i \neq y_j$.

- Delete x_i: length of LCS of $x_1 x_2 \ldots x_{i-1}$ and $y_1 y_2 \ldots y_j$.
- Delete y_j: length of LCS of $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_{j-1}$.

Bellman equation.

$$OPT(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ 1 + OPT(i-1, j-1) & \text{if } x_i = y_j \\ \max\{OPT(i-1, j),\, OPT(i, j-1)\} & \text{if } x_i \neq y_j \end{cases}$$

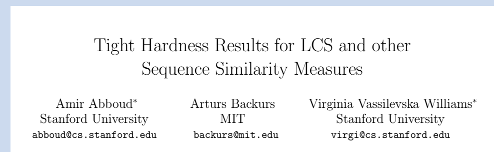
Longest common subsequence

Solution 2. Reduce to finding a min-cost alignment of x and y with

- Gap penalty $\delta = 1$
- Mismatch penalty $\alpha_{pq} = \begin{cases} 0 & \text{if } p = q \\ \infty & \text{if } p \neq q \end{cases}$
- Edit distance = # gaps = number of characters deleted from x and y.
- Length of LCS = $(m + n$ – edit distance$)$ / 2.

Analysis. $O(mn)$ time and $O(m + n)$ space.

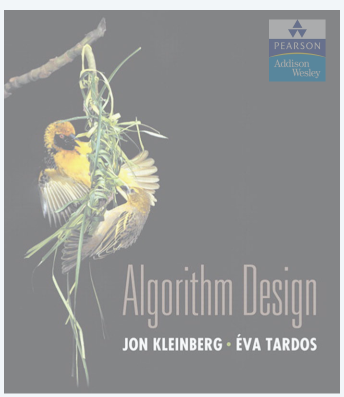
Lower bound. No $O(n^{2-\varepsilon})$ algorithm unless SETH is false.


Tight Hardness Results for LCS and other Sequence Similarity Measures

Amir Abboud* Stanford University abboud@cs.stanford.edu

Arturs Backurs MIT backurs@mit.edu

Virginia Vassilevska Williams* Stanford University virgi@cs.stanford.edu



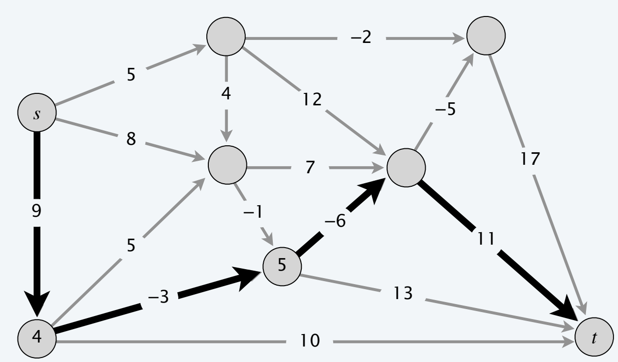
Section 6.8

6. Dynamic Programming II

Shortest paths with negative weights

Shortest-path problem. Given a digraph $G = (V, E)$, with arbitrary edge lengths ℓ_{vw}, find shortest path from source node s to destination node t.

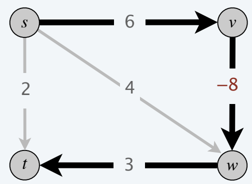
assume there exists a path from every node to t



length of shortest $s \rightsquigarrow t$ path = 9 – 3 – 6 + 11 = 11

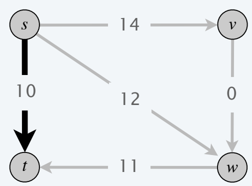
Shortest paths with negative weights: failed attempts

Dijkstra. May not produce shortest paths when edge lengths are negative.



Dijkstra selects the vertices in the order s, t, w, v
But shortest path from s to t is $s \to v \to w \to t$.

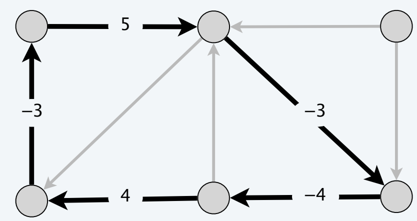
Reweighting. Adding a constant to every edge length does not necessarily make Dijkstra's algorithm produce shortest paths.



Adding 8 to each edge weight changes the shortest path from $s \to v \to w \to t$ to $s \to t$.

Negative cycles

Def. A negative cycle is a directed cycle for which the sum of its edge lengths is negative.

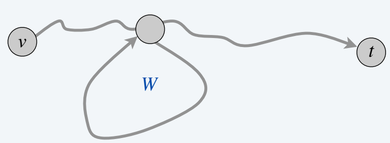


a negative cycle W : $\ell(W) \;=\; \sum_{e \in W} \ell_e \;<\; 0$

Shortest paths and negative cycles

Lemma 1. If some $v \rightsquigarrow t$ path contains a negative cycle, then there does not exist a shortest $v \rightsquigarrow t$ path.

Pf. If there exists such a cycle W, then can build a $v \rightsquigarrow t$ path of arbitrarily negative length by detouring around W as many times as desired. ▪



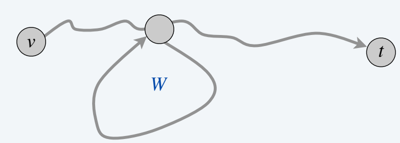
$\ell(W) < 0$

Shortest paths and negative cycles

Lemma 2. If G has no negative cycles, then there exists a shortest $v \rightsquigarrow t$ path that is simple (and has $\leq n - 1$ edges).

Pf.

- Among all shortest $v \rightsquigarrow t$ paths, consider one that uses the fewest edges.
- If that path P contains a directed cycle W, can remove the portion of P corresponding to W without increasing its length. ▪

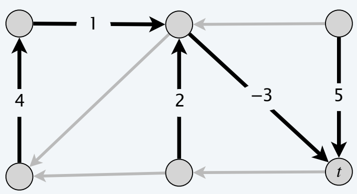


$\ell(W) \geq 0$

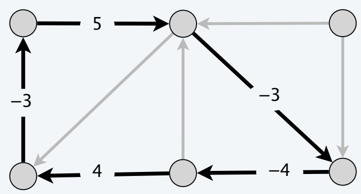
Shortest-paths and negative-cycle problems

Single-destination shortest-paths problem. Given a digraph $G = (V, E)$ with edge lengths ℓ_{vw} (but no negative cycles) and a distinguished node t, find a shortest $v \leadsto t$ path for every node v.

Negative-cycle problem. Given a digraph $G = (V, E)$ with edge lengths ℓ_{vw}, find a negative cycle (if one exists).



shortest-paths tree



negative cycle

Dynamic programming: quiz 5

Which subproblems to find shortest $v \leadsto t$ paths for every node v?

A. $OPT(i, v)$ = length of shortest $v \leadsto t$ path that uses exactly i edges.

B. $OPT(i, v)$ = length of shortest $v \leadsto t$ path that uses at most i edges.

C. Neither A nor B.

Shortest paths with negative weights: dynamic programming

Def. $OPT(i, v)$ = length of shortest $v \leadsto t$ path that uses $\leq i$ edges.

Goal. $OPT(n-1, v)$ for each v. ← by Lemma 2, if no negative cycles, there exists a shortest $v \leadsto t$ path that is simple

Case 1. Shortest $v \leadsto t$ path uses $\leq i - 1$ edges.
- $OPT(i, v) = OPT(i-1, v)$.

optimal substructure property (proof via exchange argument)

Case 2. Shortest $v \leadsto t$ path uses exactly i edges.
- if (v, w) is first edge in shortest such $v \leadsto t$ path, incur a cost of ℓ_{vw}.
- Then, select best $w \leadsto t$ path using $\leq i - 1$ edges.

Bellman equation.

$$OPT(i,v) = \begin{cases} 0 & \text{if } i = 0 \text{ and } v = t \\ \infty & \text{if } i = 0 \text{ and } v \neq t \\ \min\left\{ OPT(i-1,v),\ \min\limits_{(v,w)\in E}\{OPT(i-1,w) + \ell_{vw}\}\right\} & \text{if } i > 0 \end{cases}$$

Shortest paths with negative weights: implementation

SHORTEST-PATHS(V, E, ℓ, t)

FOREACH node $v \in V$:
  $M[0, v] \leftarrow \infty$.
$M[0, t] \leftarrow 0$.
FOR i = 1 TO $n - 1$
  FOREACH node $v \in V$:
    $M[i, v] \leftarrow M[i-1, v]$.
    FOREACH edge $(v, w) \in E$:
      $M[i, v] \leftarrow \min \{ M[i, v],\ M[i-1, w] + \ell_{vw} \}$.

Shortest paths with negative weights: implementation

Theorem 1. Given a digraph $G = (V, E)$ with no negative cycles, the DP algorithm computes the length of a shortest $v \leadsto t$ path for every node v in $\Theta(mn)$ time and $\Theta(n^2)$ space.

Pf.

- Table requires $\Theta(n^2)$ space.
- Each iteration i takes $\Theta(m)$ time since we examine each edge once. ▪

Finding the shortest paths.

- Approach 1: Maintain $successor[i, v]$ that points to next node on a shortest $v \leadsto t$ path using $\leq i$ edges.
- Approach 2: Compute optimal lengths $M[i, v]$ and consider only edges with $M[i, v] = M[i-1, w] + \ell_{vw}$. Any directed path in this subgraph is a shortest path.

Dynamic programming: quiz 6

It is easy to modify the DP algorithm for shortest paths to…

A. Compute lengths of shortest paths in $O(mn)$ time and $O(m + n)$ space.

B. Compute shortest paths in $O(mn)$ time and $O(m + n)$ space.

C. Both A and B.

D. Neither A nor B.

Shortest paths with negative weights: practical improvements

Space optimization. Maintain two 1D arrays (instead of 2D array).

- $d[v]$ = length of a shortest $v \leadsto t$ path that we have found so far.
- $successor[v]$ = next node on a $v \leadsto t$ path.

Performance optimization. If $d[w]$ was not updated in iteration $i - 1$, then no reason to consider edges entering w in iteration i.

Bellman–Ford–Moore: efficient implementation

```
BELLMAN–FORD–MOORE(V, E, c, t)

FOREACH node v ∈ V :
    d[v] ← ∞.
    successor[v] ← null.
d[t] ← 0.
FOR i = 1 TO n – 1
    FOREACH node w ∈ V :
        IF (d[w] was updated in previous pass)
            FOREACH edge (v, w) ∈ E :
                IF (d[v] > d[w] + ℓvw)
                    d[v] ← d[w] + ℓvw.
                    successor[v] ← w.
    IF (no d[·] value changed in pass i) STOP.
```

pass i
$O(m)$ time

Dynamic programming: quiz 7

Which properties must hold after pass *i* of Bellman–Ford–Moore?

A. $d[v]$ = length of a shortest $v \rightsquigarrow t$ path using $\leq i$ edges.

B. $d[v]$ = length of a shortest $v \rightsquigarrow t$ path using exactly i edges.

C. Both A and B.

D. Neither A nor B.

Bellman–Ford–Moore: analysis

Lemma 3. For each node v : $d[v]$ is the length of some $v \rightsquigarrow t$ path.
Lemma 4. For each node v : $d[v]$ is monotone non-increasing.

Lemma 5. After pass i, $d[v] \leq$ length of a shortest $v \rightsquigarrow t$ path using $\leq i$ edges.
Pf. [by induction on i]

- Base case: $i = 0$.
- Assume true after pass i.
- Let P be any $v \rightsquigarrow t$ path with $\leq i + 1$ edges.
- Let (v, w) be first edge in P and let P' be subpath from w to t.
- By inductive hypothesis, at the end of pass i, $d[w] \leq \ell(P')$ because P' is a $w \rightsquigarrow t$ path with $\leq i$ edges.
- After considering edge (v, w) in pass $i + 1$:

and by Lemma 4, $d[w]$ does not increase

$$d[v] \leq \ell_{vw} + d[w] \leq \ell_{vw} + \ell(P') = \ell(P) \quad \blacksquare$$

and by Lemma 4, $d[v]$ does not increase

Bellman–Ford–Moore: analysis

Theorem 2. Assuming no negative cycles, Bellman–Ford–Moore computes the lengths of the shortest $v \rightsquigarrow t$ paths in $O(mn)$ time and $\Theta(n)$ extra space.
Pf. Lemma 2 + Lemma 5. ▪

shortest path exists and has at most $n-1$ edges

after i passes, $d[v] \leq$ length of shortest path that uses $\leq i$ edges

Remark. Bellman–Ford–Moore is typically faster in practice.

- Edge (v, w) considered in pass $i + 1$ only if $d[w]$ updated in pass i.
- If shortest path has k edges, then algorithm finds it after $\leq k$ passes.

Dynamic programming: quiz 8

Assuming no negative cycles, which properties must hold throughout Bellman–Ford–Moore?

A. Following $successor[v]$ pointers gives a directed $v \rightsquigarrow t$ path.

B. If following $successor[v]$ pointers gives a directed $v \rightsquigarrow t$ path, then the length of that $v \rightsquigarrow t$ path is $d[v]$.

C. Both A and B.

D. Neither A nor B.

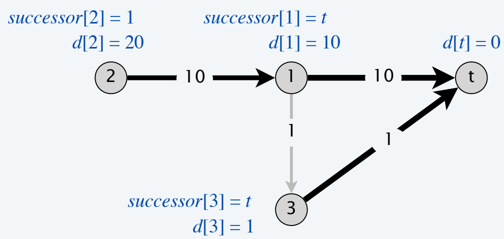
Bellman–Ford–Moore: analysis

Claim. ~~Throughout Bellman–Ford–Moore, following the *successor*[*v*] pointers gives a directed path from *v* to *t* of length *d*[*v*].~~

Counterexample. Claim is false!

- Length of successor $v \leadsto t$ path may be strictly shorter than $d[v]$.

consider nodes in order: t, 1, 2, 3



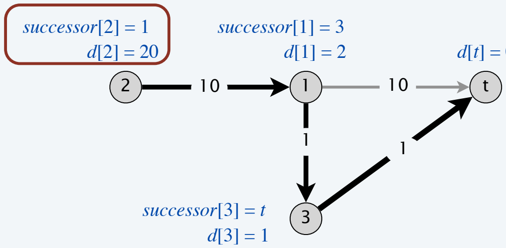
Bellman–Ford–Moore: analysis

Claim. ~~Throughout Bellman–Ford–Moore, following the *successor*[*v*] pointers gives a directed path from *v* to *t* of length *d*[*v*].~~

Counterexample. Claim is false!

- Length of successor $v \leadsto t$ path may be strictly shorter than $d[v]$.

consider nodes in order: t, 1, 2, 3



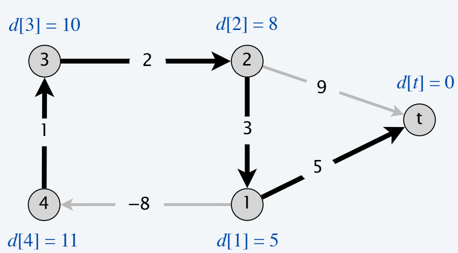
Bellman–Ford–Moore: analysis

Claim. ~~Throughout Bellman–Ford–Moore, following the *successor*[*v*] pointers gives a directed path from *v* to *t* of length *d*[*v*].~~

Counterexample. Claim is false!

- Length of successor $v \leadsto t$ path may be strictly shorter than $d[v]$.
- If negative cycle, successor graph may have directed cycles.

consider nodes in order: t, 1, 2, 3, 4



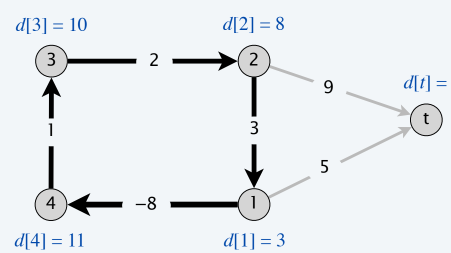
Bellman–Ford–Moore: analysis

Claim. ~~Throughout Bellman–Ford–Moore, following the *successor*[*v*] pointers gives a directed path from *v* to *t* of length *d*[*v*].~~

Counterexample. Claim is false!

- Length of successor $v \leadsto t$ path may be strictly shorter than $d[v]$.
- If negative cycle, successor graph may have directed cycles.

consider nodes in order: t, 1, 2, 3, 4



Bellman–Ford–Moore: finding the shortest paths

Lemma 6. Any directed cycle W in the successor graph is a negative cycle.

Pf.

- If $successor[v] = w$, we must have $d[v] \geq d[w] + \ell_{vw}$.
  (LHS and RHS are equal when $successor[v]$ is set; $d[w]$ can only decrease; $d[v]$ decreases only when $successor[v]$ is reset)
- Let $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k \rightarrow v_1$ be the sequence of nodes in a directed cycle W.
- Assume that (v_k, v_1) is the last edge in W added to the successor graph.
- Just prior to that:

$$\begin{aligned} d[v_1] &\geq d[v_2] + \ell(v_1, v_2) \\ d[v_2] &\geq d[v_3] + \ell(v_2, v_3) \\ &\vdots \\ d[v_{k-1}] &\geq d[v_k] + \ell(v_{k-1}, v_k) \\ d[v_k] &> d[v_1] + \ell(v_k, v_1) \end{aligned}$$

holds with strict inequality since we are updating $d[v_k]$

- Adding inequalities yields $\ell(v_1, v_2) + \ell(v_2, v_3) + \ldots + \ell(v_{k-1}, v_k) + \ell(v_k, v_1) < 0$. ▪

W is a negative cycle

Bellman–Ford–Moore: finding the shortest paths

Theorem 3. Assuming no negative cycles, Bellman–Ford–Moore finds shortest $v \leadsto t$ paths for every node v in $O(mn)$ time and $\Theta(n)$ extra space.

Pf.

- The successor graph cannot have a directed cycle. [Lemma 6]
- Thus, following the successor pointers from v yields a directed path to t.
- Let $v = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k = t$ be the nodes along this path P.
- Upon termination, if $successor[v] = w$, we must have $d[v] = d[w] + \ell_{vw}$.
  (LHS and RHS are equal when $successor[v]$ is set; $d[\cdot]$ did not change)

since algorithm terminated

- Thus,

$$\begin{aligned} d[v_1] &= d[v_2] + \ell(v_1, v_2) \\ d[v_2] &= d[v_3] + \ell(v_2, v_3) \\ &\vdots \\ d[v_{k-1}] &= d[v_k] + \ell(v_{k-1}, v_k) \end{aligned}$$

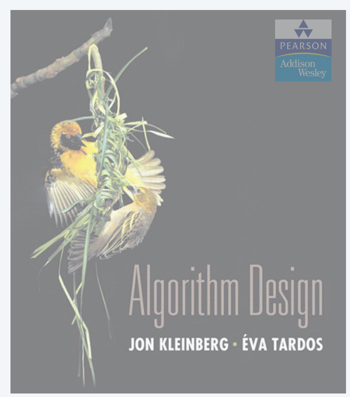
- Adding equations yields $d[v] = d[t] + \ell(v_1, v_2) + \ell(v_2, v_3) + \ldots + \ell(v_{k-1}, v_k)$. ▪

min length of any $v \leadsto t$ path (Theorem 2) — 0 — length of path P

Single-source shortest paths with negative weights

year	worst case	discovered by
1955	$O(n^4)$	Shimbel
1956	$O(m\, n^2\, W)$	Ford
1958	$O(m\, n)$	Bellman, Moore
1983	$O(n^{3/4}\, m \log W)$	Gabow
1989	$O(m\, n^{1/2} \log(nW))$	Gabow–Tarjan
1993	$O(m\, n^{1/2} \log W)$	Goldberg
2005	$O(n^{2.38}\, W)$	Sankowsi, Yuster–Zwick
2016	$\tilde{O}(n^{10/7} \log W)$	Cohen–Mądry–Sankowski–Vladu
20xx	???	

single-source shortest paths with weights between -W and W



SECTION 6.9

6. DYNAMIC PROGRAMMING II

- sequence alignment
- Hirschberg's algorithm
- Bellman–Ford–Moore algorithm
- **distance-vector protocols**
- negative cycles

Distance-vector routing protocols

Communication network.

- Node ≈ router.
- Edge ≈ direct communication link.
- Length of edge ≈ latency of link. ← non-negative, but Bellman–Ford–Moore used anyway!

Dijkstra's algorithm. Requires global information of network.

Bellman–Ford–Moore. Uses only local knowledge of neighboring nodes.

Synchronization. We don't expect routers to run in lockstep. The order in which each edges are processed in Bellman–Ford–Moore is not important. Moreover, algorithm converges even if updates are asynchronous.

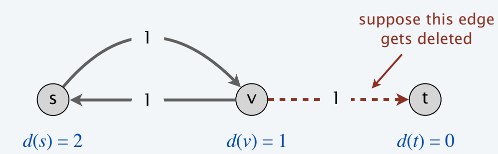
Distance-vector routing protocols

Distance-vector routing protocols. ["routing by rumor"]

- 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.

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

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



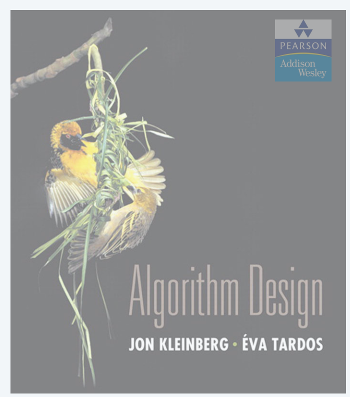
"counting to infinity"

Path-vector routing protocols

Link-state routing protocols. (not just the distance and first hop)

- Each router stores the whole path (or network topology).
- 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).



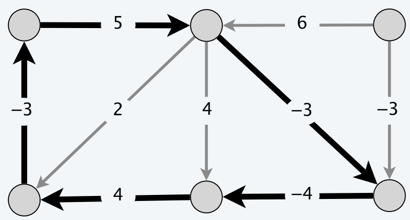
SECTION 6.10

6. DYNAMIC PROGRAMMING II

- sequence alignment
- Hirschberg's algorithm
- Bellman–Ford–Moore algorithm
- distance vector protocol
- negative cycles

Detecting negative cycles

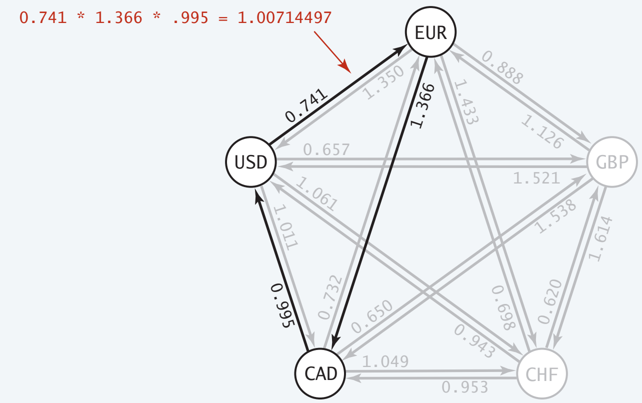
Negative cycle detection problem. Given a digraph $G = (V, E)$, with edge lengths ℓ_{vw}, find a negative cycle (if one exists).



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!



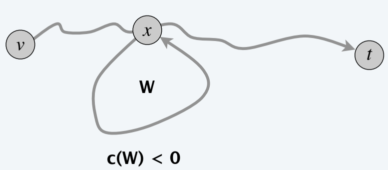
Detecting negative cycles

Lemma 7. If $OPT(n, v) = \text{OPT}(n-1, v)$ for every node v, then no negative cycles.
Pf. The $OPT(n, v)$ values have converged $\Rightarrow$ shortest $v \rightsquigarrow t$ path exists. ▪

Lemma 8. If $OPT(n, v) < \text{OPT}(n-1, v)$ for some node v, then (any) shortest $v \rightsquigarrow t$ path of length $\leq n$ contains a cycle W. Moreover W is a negative cycle.

Pf. [by contradiction]

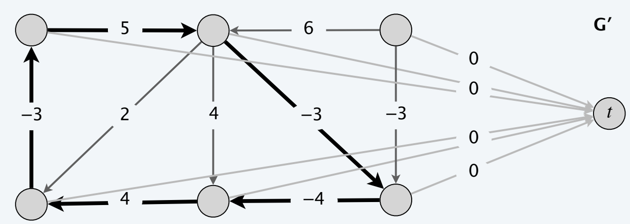
- Since $OPT(n, v) < \text{OPT}(n-1, v)$, we know that shortest $v \rightsquigarrow t$ path P has exactly n edges.
- By pigeonhole principle, the path P must contain a repeated node x.
- Let W be any cycle in P.
- Deleting W yields a $v \rightsquigarrow t$ path with $< n$ edges $\Rightarrow$ W is a negative cycle. ▪



Detecting negative cycles

Theorem 4. Can find a negative cycle in $\Theta(mn)$ time and $\Theta(n^2)$ space.
Pf.

- Add new sink node t and connect all nodes to t with 0-length edge.
- G has a negative cycle iff G' has a negative cycle.
- Case 1. [$OPT(n, v) = OPT(n-1, v)$ for every node v]
  By Lemma 7, no negative cycles.
- Case 2. [$OPT(n, v) < OPT(n-1, v)$ for some node v]
  Using proof of Lemma 8, can extract negative cycle from $v \rightsquigarrow t$ path.
  (cycle cannot contain t since no edge leaves t) ▪



Detecting negative cycles

Theorem 5. Can find a negative cycle in $O(mn)$ time and $O(n)$ extra space.

Pf.

- Run Bellman–Ford–Moore on G' for $n' = n + 1$ passes (instead of $n' - 1$).
- If no $d[v]$ values updated in pass n', then no negative cycles.
- Otherwise, suppose $d[s]$ updated in pass n'.
- Define $pass(v)$ = last pass in which $d[v]$ was updated.
- Observe $pass(s) = n'$ and $pass(successor[v]) \geq pass(v) - 1$ for each v.
- Following successor pointers, we must eventually repeat a node.
- Lemma 6 $\Rightarrow$ the corresponding cycle is a negative cycle. ▪

Remark. See p. 304 for improved version and early termination rule. (Tarjan's subtree disassembly trick)

Dynamic programming: quiz 9

How difficult to find a negative cycle in an undirected graph?

A. $O(m \log n)$

B. $O(mn)$

C. $O(mn + n^2 \log n)$

D. $O(n^{2.38})$

E. No poly-time algorithm is known.