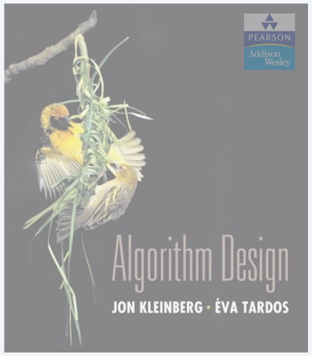
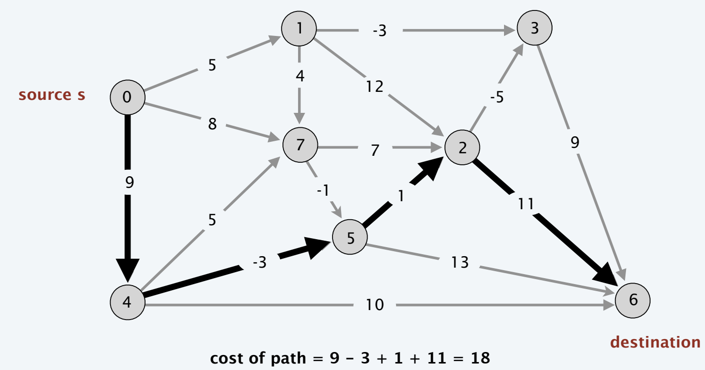


6. Dynamic Programming II

- sequence alignment
- Hirschberg's algorithm
- **Bellman-Ford**
- distance vector protocols
- negative cycles in a digraph

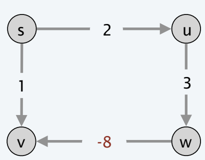
Shortest paths

Shortest path problem. Given a digraph $G = (V, E)$, with arbitrary edge weights or costs c_{vw}, find cheapest path from node s to node t.

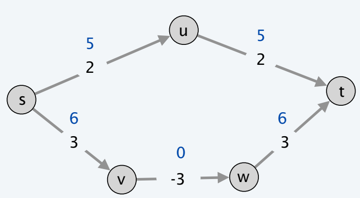


Shortest paths: failed attempts

Dijkstra. Can fail if negative edge weights.

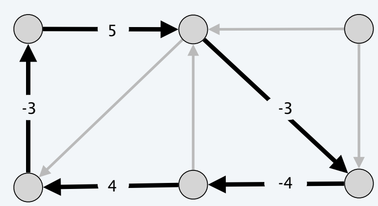


Reweighting. Adding a constant to every edge weight can fail.



Negative cycles

Def. A negative cycle is a directed cycle such that the sum of its edge weights is negative.

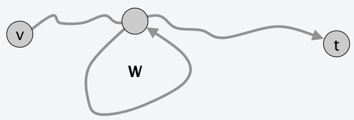


a negative cycle W : $c(W) \;=\; \sum_{e \in W} c_e \;<\; 0$

Shortest paths and negative cycles

Lemma 1. If some path from v to t contains a negative cycle, then there does not exist a cheapest path from v to t.

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



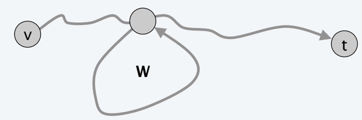
c(W) < 0

Shortest paths and negative cycles

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

Pf.

- Consider a cheapest $v \rightsquigarrow t$ path P that uses the fewest number of edges.
- If P contains a cycle W, can remove portion of P corresponding to W without increasing the cost. ▪

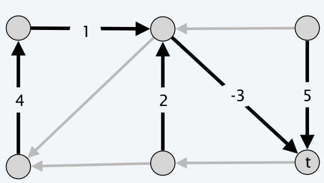


c(W) ≥ 0

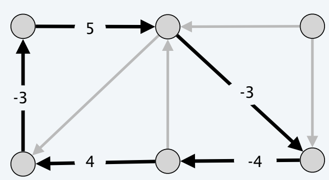
Shortest path and negative cycle problems

Shortest path problem. Given a digraph $G = (V, E)$ with edge weights c_{vw} and no negative cycles, find cheapest $v \rightsquigarrow t$ path for each node v.

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



shortest-paths tree



negative cycle

Shortest paths: dynamic programming

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

- Case 1: Cheapest $v \rightsquigarrow t$ path uses $\leq i - 1$ edges.
  - $OPT(i, v) = \text{OPT}(i - 1, v)$

optimal substructure property
(proof via exchange argument)

- Case 2: Cheapest $v \rightsquigarrow t$ path uses exactly i edges.
  - if (v, w) is first edge, then OPT uses (v, w), and then selects best $w \rightsquigarrow t$ path using $\leq i - 1$ edges

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

Observation. If no negative cycles, $OPT(n - 1, v)$ = cost of cheapest $v \rightsquigarrow t$ path.

Pf. By Lemma 2, cheapest $v \rightsquigarrow t$ path is simple. ▪

Shortest paths: implementation

```
SHORTEST-PATHS (V, E, c, 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_vw }.
```


Shortest paths: implementation

Theorem 1. Given a digraph $G = (V, E)$ with no negative cycles, the dynamic programming algorithm computes the cost of the cheapest $v \rightsquigarrow t$ path for each 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 a $successor(i, v)$ that points to next node on cheapest $v \rightsquigarrow t$ path using at most i edges.
- Approach 2: Compute optimal costs $M[i, v]$ and consider only edges with $M[i, v] = M[i - 1, w] + c_{vw}$.

Shortest paths: practical improvements

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

- $d(v)$ = cost of cheapest $v \rightsquigarrow t$ path that we have found so far.
- $successor(v)$ = next node on a $v \rightsquigarrow t$ 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: efficient implementation

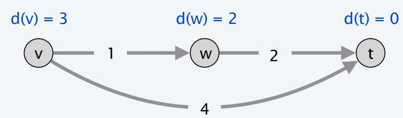
```
BELLMAN-FORD (V, E, c, t)

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 iteration)      │
         FOREACH edge (v, w) ∈ E                       │ 1 pass
            IF ( d(v) > d(w) + c_vw)                   │
               d(v) ← d(w) + c_vw.                     │
               successor(v) ← w.                       ┘
   IF no d(w) value changed in iteration i, STOP.
```

Bellman-Ford: analysis

Claim. ~~After the i^{th} pass of Bellman-Ford, $d(v)$ equals the cost of the cheapest $v \to t$ path using at most i edges~~.

Counterexample. Claim is false!



if nodes w considered before node v, then d(v) = 3 after 1 pass

Bellman-Ford: analysis

Lemma 3. Throughout Bellman-Ford algorithm, $d(v)$ is the cost of some $v \to t$ path; after the i^{th} pass, $d(v)$ is no larger than the cost of the cheapest $v \to t$ path using $\leq i$ edges.

Pf. [by induction on i]

- Assume true after i^{th} pass.
- Let P be any $v \to t$ path with $i + 1$ edges.
- Let (v, w) be first edge on path and let P' be subpath from w to t.
- By inductive hypothesis, $d(w) \leq c(P')$ since P' is a $w \to t$ path with i edges.
- After considering v in pass $i+1$: $d(v) \leq c_{vw} + d(w)$

$$\begin{aligned} d(v) &\leq c_{vw} + d(w) \\ &\leq c_{vw} + c(P') \\ &= c(P) \quad \blacksquare \end{aligned}$$

Theorem 2. Given a digraph with no negative cycles, Bellman-Ford computes the costs of the cheapest $v \to t$ paths in $O(mn)$ time and $\Theta(n)$ extra space.

can be substantially faster in practice

Pf. Lemmas 2 + 3. ▪

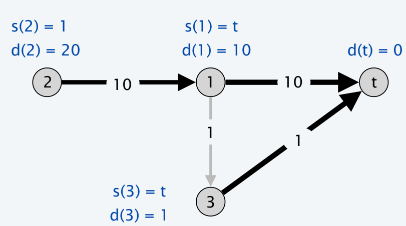
Bellman-Ford: analysis

Claim. ~~Throughout the Bellman-Ford algorithm, following $successor(v)$ pointers gives a directed path from v to t of cost $d(v)$.~~

Counterexample. Claim is false!

- Cost of successor $v \to t$ path may have strictly lower cost than $d(v)$.

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



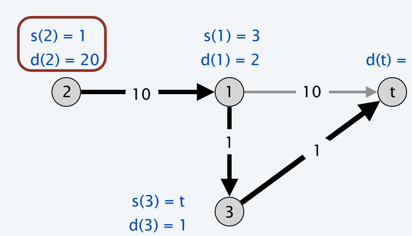
Bellman-Ford: analysis

Claim. ~~Throughout the Bellman-Ford algorithm, following $successor(v)$ pointers gives a directed path from v to t of cost $d(v)$.~~

Counterexample. Claim is false!

- Cost of successor $v \to t$ path may have strictly lower cost than $d(v)$.

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



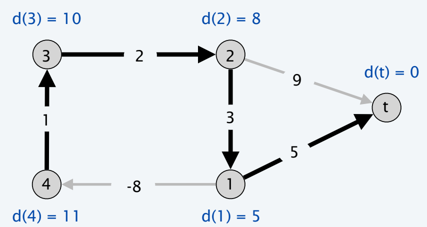
Bellman-Ford: analysis

Claim. ~~Throughout the Bellman-Ford algorithm, following $successor(v)$ pointers gives a directed path from v to t of cost $d(v)$.~~

Counterexample. Claim is false!

- Cost of successor $v \rightsquigarrow t$ path may have strictly lower cost than $d(v)$.
- Successor graph may have cycles.

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



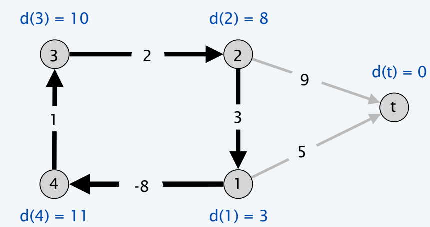
Bellman-Ford: analysis

Claim. ~~Throughout the Bellman-Ford algorithm, following $successor(v)$ pointers gives a directed path from v to t of cost $d(v)$.~~

Counterexample. Claim is false!

- Cost of successor $v \rightsquigarrow t$ path may have strictly lower cost than $d(v)$.
- Successor graph may have cycles.

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



Bellman-Ford: finding the shortest path

Lemma 4. If the successor graph contains a directed cycle W, then W is a negative cycle.

Pf.

- If $successor(v) = w$, we must have $d(v) \geq d(w) + c_{vw}$.
  (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 $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ be the nodes along the cycle W.
- Assume that (v_k, v_1) is the last edge added to the successor graph.
- Just prior to that:

$$\begin{array}{lcll} d(v_1) & \geq & d(v_2) & +\ c(v_1, v_2) \\ d(v_2) & \geq & d(v_3) & +\ c(v_2, v_3) \\ \vdots & \vdots & & \vdots \\ d(v_{k-1}) & \geq & d(v_k) & +\ c(v_{k-1}, v_k) \\ d(v_k) & > & d(v_1) & +\ c(v_k, v_1) \end{array}$$

holds with strict inequality since we are updating $d(v_k)$

- Adding inequalities yields $\underbrace{c(v_1, v_2) + c(v_2, v_3) + \ldots + c(v_{k-1}, v_k) + c(v_k, v_1) < 0}_{\text{W is a negative cycle}}$. ▪

Bellman-Ford: finding the shortest path

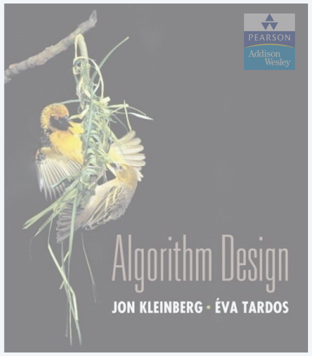
Theorem 3. Given a digraph with no negative cycles, Bellman-Ford finds the cheapest $s \rightsquigarrow t$ paths in $O(mn)$ time and $\Theta(n)$ extra space.

Pf.

- The successor graph cannot have a negative cycle. [Lemma 4]
- Thus, following the successor pointers from s yields a directed path to t.
- Let $s = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k = t$ be the nodes along this path P.
- Upon termination, if $successor(v) = w$, we must have $d(v) = d(w) + c_{vw}$.
  (LHS and RHS are equal when $successor(v)$ is set; $d(\cdot)$ did not change)
  since algorithm terminated
- Thus,

$$\begin{array}{lcll} d(v_1) & = & d(v_2) & +\ c(v_1, v_2) \\ d(v_2) & = & d(v_3) & +\ c(v_2, v_3) \\ \vdots & \vdots & & \vdots \\ d(v_{k-1}) & = & d(v_k) & +\ c(v_{k-1}, v_k) \end{array}$$

- Adding equations yields $\underbrace{d(s)}_{\substack{\text{min cost} \\ \text{of any s}\rightsquigarrow\text{t path} \\ \text{(Theorem 2)}}} = \underbrace{d(t)}_{0} + \underbrace{c(v_1, v_2) + c(v_2, v_3) + \ldots + c(v_{k-1}, v_k)}_{\text{cost of path P}}$. ▪



6. Dynamic Programming II

- sequence alignment
- Hirschberg's algorithm
- Bellman-Ford
- *distance vector protocols*
- negative cycles in a digraph

Distance vector protocols

Communication network.

- Node ≈ router.
- Edge ≈ direct communication link.
- Cost of edge ≈ 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 in not important. Moreover, algorithm still converges even if updates are asynchronous.

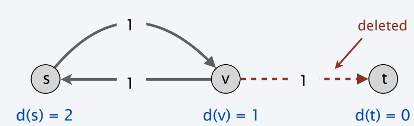
Distance vector protocols

Distance vector 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 costs may change during algorithm (or fail completely).



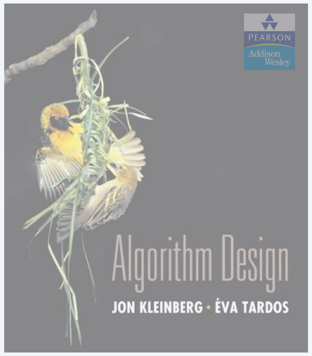
"counting to infinity"

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).

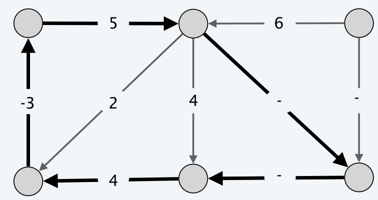


6. Dynamic Programming II

- *sequence alignment*
- *Hirschberg's algorithm*
- *Bellman-Ford*
- *distance vector protocol*
- *negative cycles in a digraph*

Detecting negative cycles

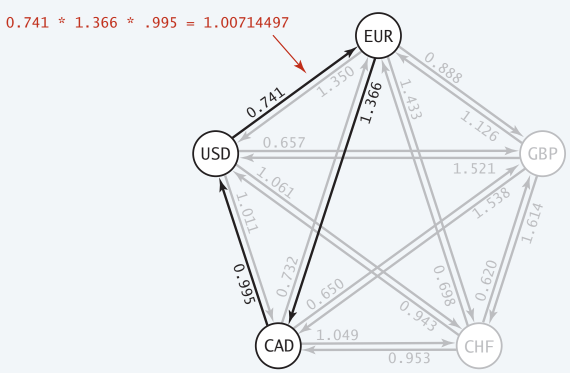
Negative cycle detection problem. Given a digraph $G = (V, E)$, with edge weights c_{vw}, find a negative cycle (if one exists).



Detecting negative cycles: application

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

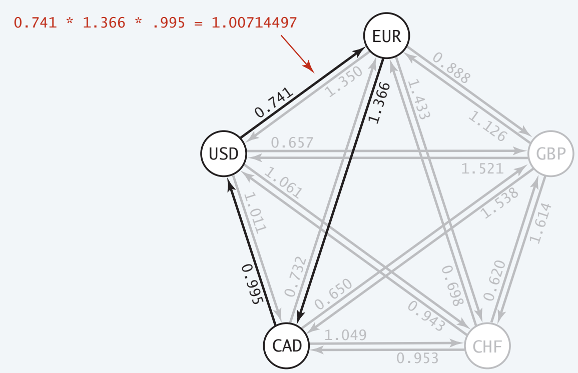
Remark. Fastest algorithm very valuable!



Arbitrage opportunities

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

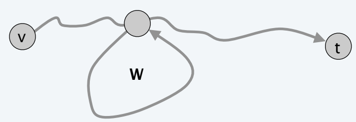
Lemma 5. If $OPT(n, v) = \text{OPT}(n-1, v)$ for all v, then no negative cycle can reach t.

Pf. Bellman-Ford algorithm. ▪

Lemma 6. If $OPT(n, v) < \text{OPT}(n-1, v)$ for some node v, then (any) cheapest path from v to t 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, P must contain a directed cycle W.
- Deleting W yields a $v \rightsquigarrow t$ path with $< n$ edges $\Rightarrow$ W has negative cost. ▪



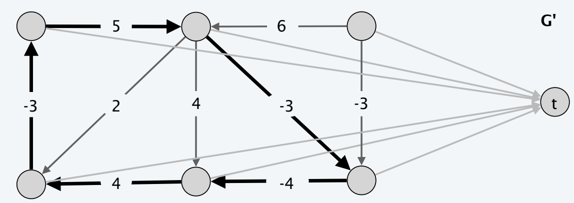
c(W) < 0

Detecting negative cycles

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

Pf.

- Add new node t and connect all nodes to t with 0-cost edge.
- G has a negative cycle iff G' has a negative cycle than can reach t.
- If $OPT(n, v) = OPT(n-1, v)$ for all nodes v, then no negative cycles.
- If not, then extract directed cycle from path from v to t. (cycle cannot contain t since no edges leave t) ▪



Detecting negative cycles

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

Pf.

- Run Bellman-Ford for n passes (instead of $n-1$) on modified digraph.
- 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 4 $\Rightarrow$ this cycle is a negative cycle. ▪

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