Algorithms: Dynamic Programming

Shortest Path Problems: Floyd-Warshall Algorithm Johnson's Algorithm

Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET

Shortest-Path Problems

- Shortest-Path problems
  - **Single-Source (Single-Destination):** Find a shortest path from a given source (vertex *s*) to each of the vertices.
  - **Single-Pair:** Given two vertices, find a shortest path between them. Solution to single-source problem solves this problem efficiently, too.
  - **All-Pairs:** Find shortest-paths for every pair of vertices. Dynamic programming algorithm.

All-Pairs Shortest Paths

- We want to compute a table giving the length of the shortest path between any two vertices. (We also would like to get the shortest paths themselves.)
- We could just call Dijkstra or Bellman-Ford |V| times, passing a different source vertex each time.
- It can be done in $\Theta(V^3)$, which seems to be as good as you can do on dense graphs.

Doing APSP with SSSP

- Dijkstra would take time

  $\Theta(V \times E \lg V) = \Theta(VE \lg V)$ [by using ordinary heaps]

  $\Theta(V \times (V \lg V + E)) = \Theta(V^2 \lg V + VE)$ [Fibonacci heaps],

  but doesn't work with negative-weight edges.

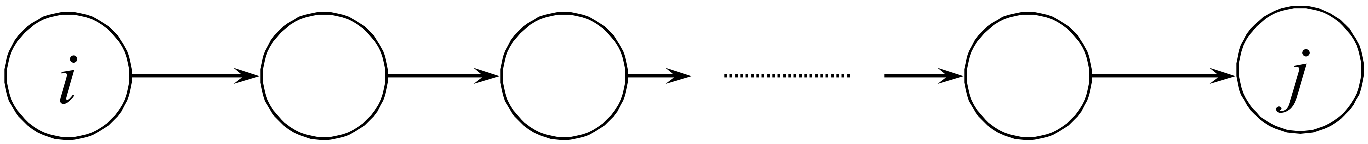
- Bellman-Ford would take $\Theta(V \times VE) = \Theta(V^2 E)$.

The Floyd-Warshall Algorithm

- Represent the directed, edge-weighted graph in adjacency-matrix form.
- W= matrix of weights = $\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$
  - w_{ij} is the weight of edge (i, j), or ∞ if there is no such edge.
  - Return a matrix D, where each entry d_{ij} is $\delta(i, j)$.
  - Could also return a predecessor matrix, Π, where each entry π_{ij} is the predecessor of j on the shortest path from i.

Floyd-Warshall: Idea

- Consider *intermediate vertices* of a path:

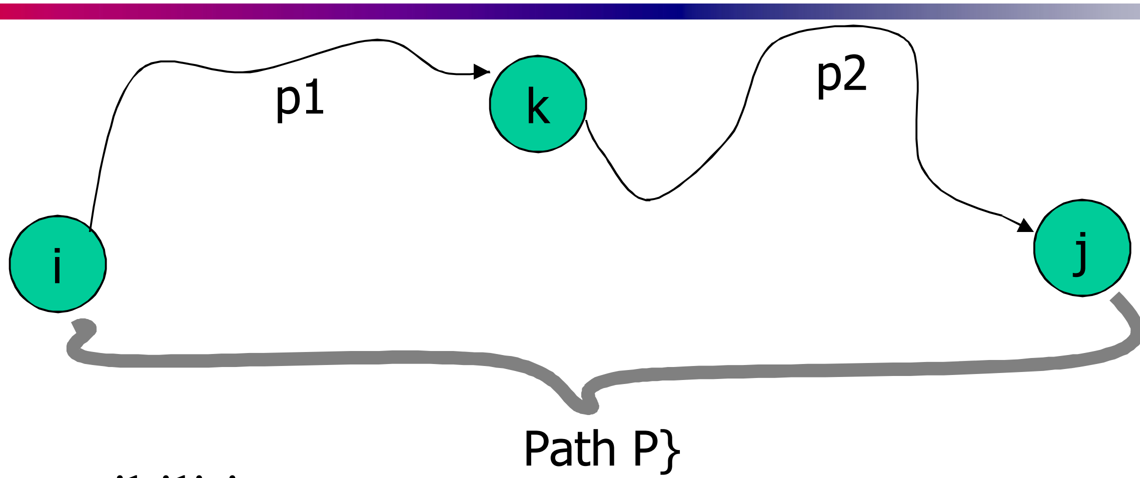


Say we know the length of the shortest path from i to j whose intermediate vertices are only those with numbers 1, 2, ..., k-1. Call this length $d_{ij}^{(k-1)}$

Now how can we extend this from k-1 to k ? In other words, we want to compute $d_{ij}^{(k)}$.

Can we use $d_{ij}^{(k-1)}$, and if so how ?

Floyd-Warshall: Idea



Two possibilities:

1. *k* is not an intermediate vertex on *P:*

   the path through vertices 1 ... *k*-1 is still the shortest.

2. *k* is an intermediate vertex on *P:*

   there is a shorter path consisting of two subpaths, one from *i* to *k* and one from *k* to *j*.

If the vertex k is not an intermediate vertex on P, then

$$d_{ij}^{(k)} = d_{ij}^{(k-1)}$$

If the vertex k is an intermediate vertex on P, then

$$d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$$

Floyd-Warshall: Idea

Therefore, we can conclude that

$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$$

If we do not use intermediate nodes, i.e., when k=0, then

$$d_{ij}^{(0)} = w_{ij}$$

If k > 0, then

$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$$

When $k = |V|$, we're done.

Dynamic Programming

- Floyd-Warshall is a *dynamic programming* algorithm:
- Compute and store solutions to sub-problems. Combine those solutions to solve larger sub-problems.
- Here, the sub-problems involve finding the shortest paths through a subset of the vertices.

Code for Floyd-Warshall

Floyd-Warshall(W)

$n \leftarrow rows[W]$ // number of vertices

$D^{(0)} \leftarrow W$

for $k \leftarrow 1$ to n do

  for i $\leftarrow 1$ to n do

    for $j \leftarrow 1$ to n do

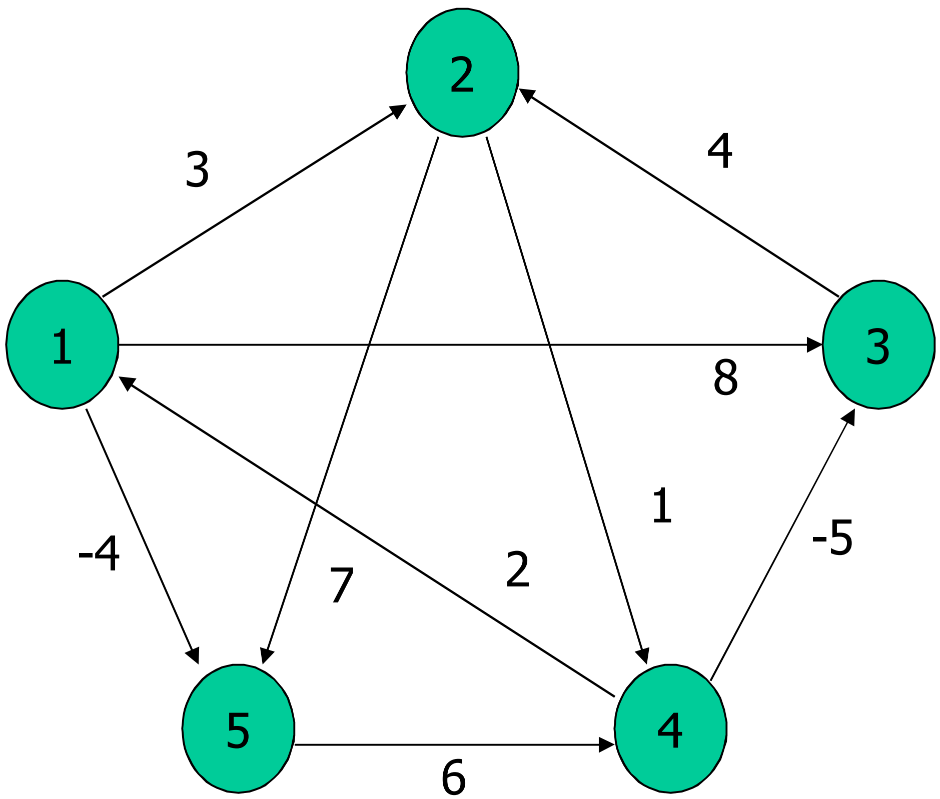
      $d_{ij}^{(k)} \leftarrow \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$

return $D^{(n)}$

Running time: $\theta(V^3)$.

(Small constant, because operations are simple.)

Example of Floyd-Warshall



$$D(0)=\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D(1)=\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \mathbf{5} & -5 & 0 & -\mathbf{2} \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

Example of Floyd-Warshall

$$D^{(0)}=\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(0)}=\begin{pmatrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & NIL & NIL \\ 4 & NIL & 4 & NIL & NIL \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

$$D^{(1)}=\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(1)}=\begin{pmatrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & NIL & NIL \\ 4 & 1 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

Example of Floyd-Warshall

D(2)= $\begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$

$\prod$(2)= $\begin{pmatrix} NIL & 1 & 1 & 2 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 2 & 2 \\ 4 & 1 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$

D(3)= $\begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$

$\prod$(3)= $\begin{pmatrix} NIL & 1 & 1 & 2 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 2 & 2 \\ 4 & 3 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$

Example of Floyd-Warshall

$$D^{(4)}= \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$\prod^{(4)}= \begin{pmatrix} NIL & 1 & 4 & 2 & 1 \\ 4 & NIL & 4 & 2 & 1 \\ 4 & 3 & NIL & 2 & 1 \\ 4 & 3 & 4 & NIL & 1 \\ 4 & 3 & 4 & 5 & NIL \end{pmatrix}$$

$$D^{(5)}= \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$\prod^{(5)}= \begin{pmatrix} NIL & 3 & 4 & 5 & 1 \\ 4 & NIL & 4 & 2 & 1 \\ 4 & 3 & NIL & 2 & 1 \\ 4 & 3 & 4 & NIL & 1 \\ 4 & 3 & 4 & 5 & NIL \end{pmatrix}$$

Johnson's Algorithm

- Makes clever use of Bellman-Ford and Dijkstra to do All-Pairs-Shortest-Paths efficiently on sparse graphs
- Motivation: By running Dijkstra $|V|$ times, we could do APSP in time
  - $\Theta(V^2 \lg V + VE \lg V)$ (Modified Dijkstra), or
  - $\Theta(V^2 \lg V + VE)$ (Fibonacci Dijkstra).

  This beats $\Theta(V^3)$ (Floyd-Warshall) when the graph is sparse
- Problem: negative edge weights

The Basic Idea

- Reweight the edges so that:
  - No edge weight is negative.
  - Shortest paths are preserved (A shortest path in the original graph is still one in the new, reweighted graph)

- An obvious attempt: subtract the minimum weight from all the edge weights.

  For example, if the minimum weight is -2:

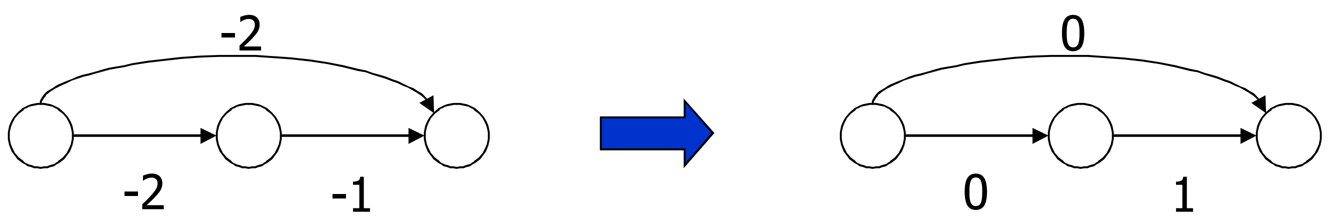
  $$-2 - (-2) = 0$$
  $$3 - (-2) = 5$$

  etc.

Counter Example

- Subtracting the minimum weight from every weight doesn't work.

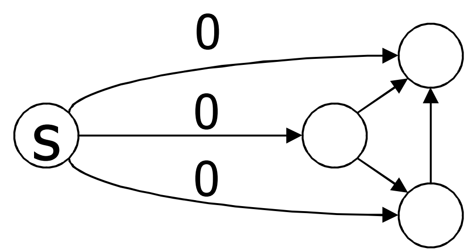
  Consider:



- Paths with more edges are unfairly penalized.

Johnson's Insight

- Add a vertex s to the original graph G, with edges of weight 0 to each vertex in G:



- Assign new weights $\hat{w}$ to each edge as follows:

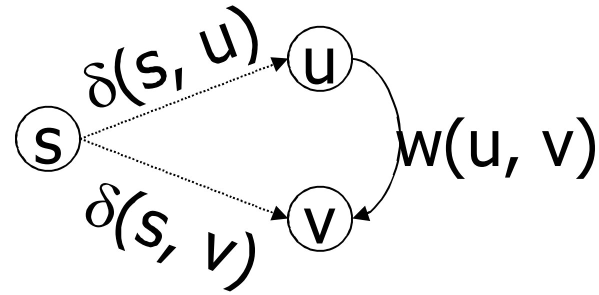
$$\hat{w}(u, v) = w(u, v) + \delta(s, u) - \delta(s, v)$$

Question 1

- Are all the ŵ's non-negative? **Yes:**

$$\delta(s,u) + w(u,v) \text{ must be} \geq \delta(s,v)$$

Otherwise, s ⇒ u → v would be shorter than the shortest path from s to v.



$$\delta(s,u) + w(u,v) \geq \delta(s,v)$$

Rewriting :

$$\underbrace{w(u,v) + \delta(s,u) - \delta(s,v)}_{\hat{w}(u,v)} \geq 0$$

Question 2

- Does the reweighting preserve shortest paths? **Yes:**

Consider any path $p = v_1, v_2, \ldots, v_k$

$$
\begin{aligned}
\hat{w}(p) &= \sum_{i=1}^{k-1} \hat{w}(v_i, v_{i+1}) \\
&= w(v_1, v_2) + \delta(s, v_1) - \delta(s, v_2) \\
&+ w(v_2, v_3) \qquad\qquad + \delta(s, v_2) - \delta(s, v_3) \\
&\quad\vdots \\
&+ w(v_{k-1}, v_k) \qquad\qquad + \delta(s, v_{k-1}) - \delta(s, v_k) \\
\hline
&= w(p) + \underbrace{\delta(s, v_1) - \delta(s, v_k)}_{\text{A value that depends only on the endpoints, not on the path.}}
\end{aligned}
$$

Because $\delta(s, v_1)$ and $\delta(s, v_k)$ do not depend on the path, if one path from v_1 to v_k is shorter than another using weight function w, then it is also shorter using $\hat{w}$. Thus, shortest paths will be preserved.

Question 3

- How do we compute the $\delta(s, v)$'s?

  Use Bellman-Ford Algorithm.

  This also tells us if we have a negative-weight cycle.

Johnson's Algorithm

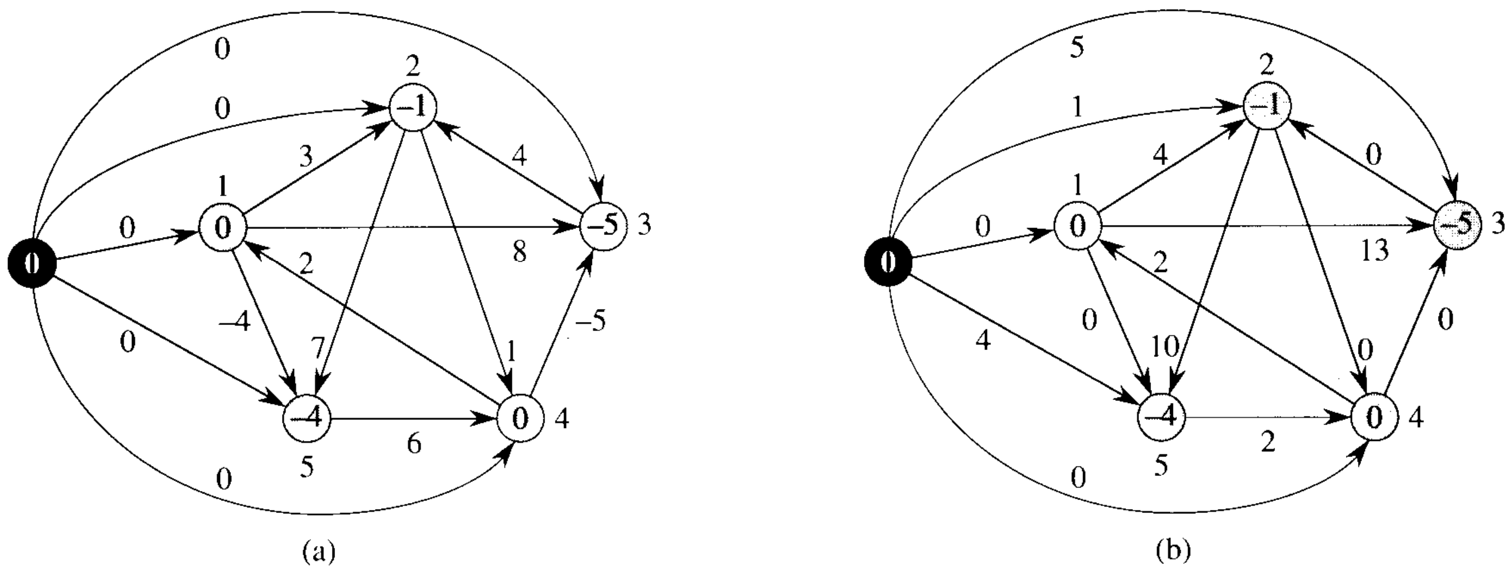
1. Compute G', which consists of G augmented with s and a zero-weight edge from s to every vertex in G.
2. Run Bellman-Ford(G', w, s) to obtain the $\delta(s, v)$'s
3. Reweight by computing $\hat{w}$ for each edge
4. Run Dijkstra on each vertex to compute δ'
5. Undo reweighting factors to compute δ

Johnson's Algorithm: CLRS

JOHNSON(G)

compute G', where $V[G'] = V[G] \cup \{s\}$,
    $E[G'] = E[G] \cup \{(s, v) : v \in V[G]\}$, and
    $w(s, v) = 0$ for all $v \in V[G]$
if BELLMAN-FORD$(G', w, s) =$ FALSE
  then print "the input graph contains a negative-weight cycle"
  else **for** each vertex $v \in V[G']$
      do set $h(v)$ to the value of $\delta(s, v)$
        computed by the Bellman-Ford algorithm
    for each edge $(u, v) \in E[G']$
      do $\widehat{w}(u, v) \leftarrow w(u, v) + h(u) - h(v)$
    for each vertex $u \in V[G]$
      do run DIJKSTRA$(G, \widehat{w}, u)$ to compute $\widehat{\delta}(u, v)$ for all $v \in V[G]$
        for each vertex $v \in V[G]$
          do $d_{uv} \leftarrow \widehat{\delta}(u, v) + h(v) - h(u)$
    return D

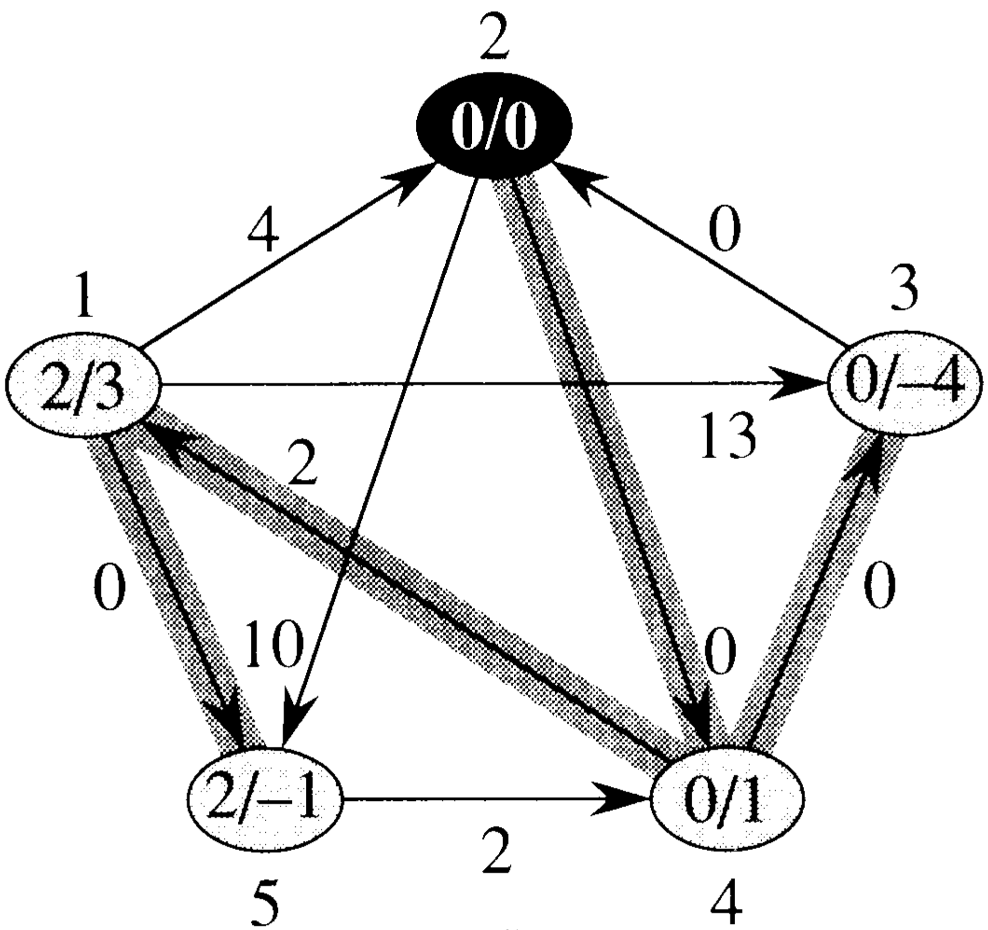
Johnson: reweighting



(a) (b)

$$\hat{w}(u, v) = w(u, v) + \delta(s, u) - \delta(s, v)$$

Johnson using Dijkstra



(d)

Johnson's: Running Time

1. Computing G′: $\Theta(V)$
2. Bellman-Ford: $\Theta(VE)$
3. Reweighting: $\Theta(E)$
4. Running (Modified) Dijkstra: $\Theta(V^2 \lg V + VE \lg V)$
5. Adjusting distances: $\Theta(V^2)$

Total is dominated by Dijkstra: $\Theta(V^2 \lg V + VE \lg V)$