Shortest Path Problems II

AU4606: Network Optimization

AI4702: Network Intelligence and Optimization

Xiaoming Duan
Department of Automation
Shanghai Jiao Tong University

September 28, 2023

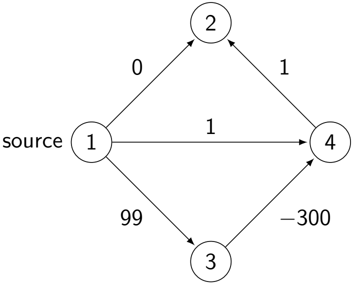
Last time

- Shortest path problems: formulation
  - What is a shortest path problem
  - An "unusual" application
- Algorithms for shortest path problems
  - Two properties
  - Algorithms for acyclic graphs (pulling and reaching)
  - Dijkstra's algorithm

Today

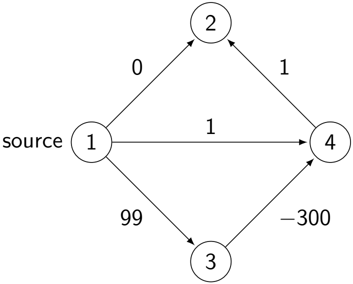
1. Shortest path problems with negative weights
   - Issues with Dijkstra's algorithm
   - Optimality condition
   - Label-correcting algorithm
   - Bellman-Ford algorithm
   - All-pair shortest path problems

How does Dijkstra's algorithm fail?



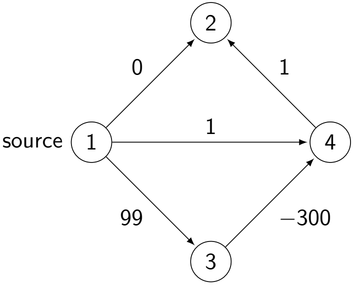
1. Initialization: $d(1) = 0$

How does Dijkstra's algorithm fail?



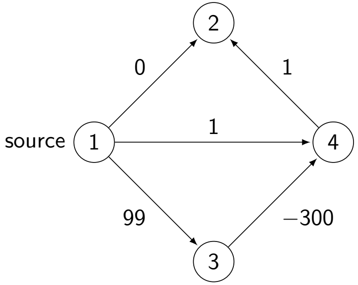
1. Initialization: $d(1) = 0$
2. Pick node 1: $d(2) = 0$, $d(3) = 99$, $d(4) = 1$

How does Dijkstra's algorithm fail?



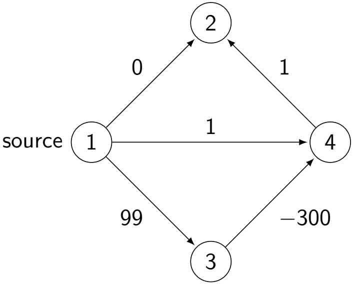
1. Initialization: $d(1) = 0$
2. Pick node 1: $d(2) = 0$, $d(3) = 99$, $d(4) = 1$
3. Pick node 2: none

How does Dijkstra's algorithm fail?



1. Initialization: $d(1) = 0$
2. Pick node 1: $d(2) = 0$, $d(3) = 99$, $d(4) = 1$
3. Pick node 2: none
4. Pick node 4: none

How does Dijkstra's algorithm fail?



1. Initialization: $d(1) = 0$
2. Pick node 1: $d(2) = 0$, $d(3) = 99$, $d(4) = 1$
3. Pick node 2: none
4. Pick node 4: none
5. Pick node 3: $d(4) = -201$, terminate

Not able to find the shortest path $1 - 3 - 4 - 2$ to node 2

Why does Dijkstra's algorithm fail?

Correctness of Dikstra algorithm

Given a directed network $G = (N, A)$ with nonnegative arc costs, Dijkstra's algorithm correctly determines the shortest path distances from the source s to every node in the network.

- We proved that when a node is selected, its distance label is optimal
  1. Base case: distance label of source is optimal
  2. Ind. hypothesis: distance labels of first $k - 1$ selected nodes are optimal
  3. Ind. step: by contradiction where nonnegativity of arc lengths is crucial
- The optimality of distance labels of selected node is not guaranteed

New algorithms are needed.

Do the two properties of shortest path problems still hold?

Subpaths are shortest

If the path $s = i_l - i_2 - \cdots - i_h = k$ is a shortest path from node s to node k, then for every $q \in \{2, 3, ..., h-1\}$, the subpath $s = i_l - i_2 - \cdots - i_q$ is a shortest path from the source node to node i_q.

Distance labels

Let the vector d represent the shortest path distances. Then a directed path P from the source node to node k is a shortest path if and only if $d(j) = d(i) + c_{ij}$ for every arc $(i, j) \in P$.

Do the two properties of shortest path problems still hold?

Subpaths are shortest

If the path $s = i_l - i_2 - \cdots - i_h = k$ is a shortest path from node s to node k, then for every $q \in \{2, 3, ..., h-1\}$, the subpath $s = i_l - i_2 - \cdots - i_q$ is a shortest path from the source node to node i_q.

Distance labels

Let the vector d represent the shortest path distances. Then a directed path P from the source node to node k is a shortest path if and only if $d(j) = d(i) + c_{ij}$ for every arc $(i, j) \in P$.

These properties still hold so long as there are no negative cycles.

Optimality condition

Distance labels

Let the vector d represent the shortest path distances. Then a directed path P from the source node to node k is a shortest path if and only if $d(j) = d(i) + c_{ij}$ for every arc $(i, j) \in P$.

Optimality condition

Distance labels

Let the vector d represent the shortest path distances. Then a directed path P from the source node to node k is a shortest path if and only if $d(j) = d(i) + c_{ij}$ for every arc $(i,j) \in P$.

Shortest path optimality condition

For each $k \in N$, let $d(k)$ be the length of some directed path from the source to node k. Then the numbers d represents the shortest path distances if and only if they satisfy the following shortest path optimality conditions:

$$d(j) \leq d(i) + c_{ij} \qquad \text{for all } (i,j) \in A.$$

Optimality condition

Distance labels

Let the vector d represent the shortest path distances. Then a directed path P from the source node to node k is a shortest path if and only if $d(j) = d(i) + c_{ij}$ for every arc $(i, j) \in P$.

Shortest path optimality condition

For each $k \in N$, let $d(k)$ be the length of some directed path from the source to node k. Then the numbers d represents the shortest path distances if and only if they satisfy the following shortest path optimality conditions:

$$d(j) \leq d(i) + c_{ij} \qquad \text{for all } (i, j) \in A.$$

These optimality conditions suggest a very natural algorithm.

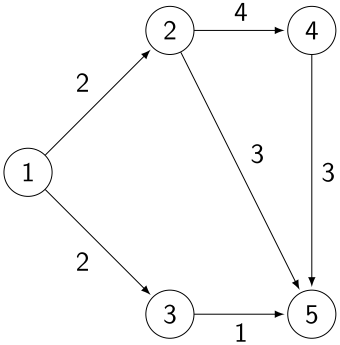
Generic label-correcting algorithm

Algorithm Label-correcting

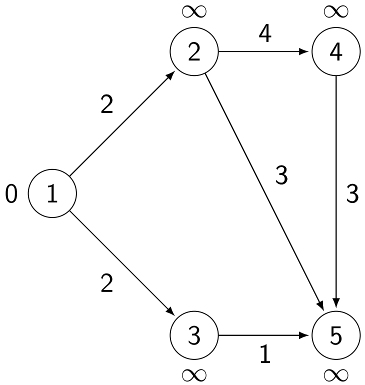
$d(s) \leftarrow 0$ and $\text{pred}(s) \leftarrow 0$
$d(j) \leftarrow \infty$ for each node $j \in N \setminus \{s\}$
while some arc (i, j) satisfies $d(j) > d(i) + c_{ij}$ **do**
    $d(j) \leftarrow d(i) + c_{ij}$
    $\text{pred}(j) \leftarrow i$
end while

- "Label-setting" vs "label-correcting"

Label-correcting algorithm: running example

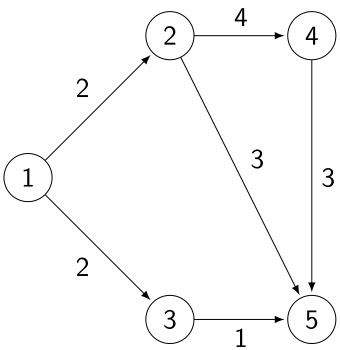


(a) A directed graph

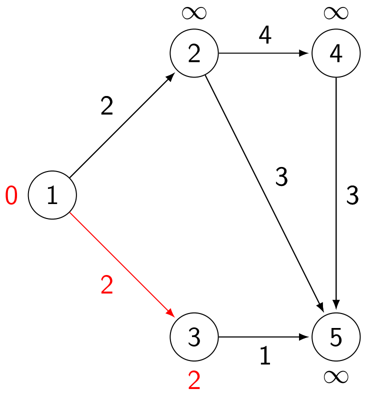


(b) Label-correcting

Label-correcting algorithm: running example



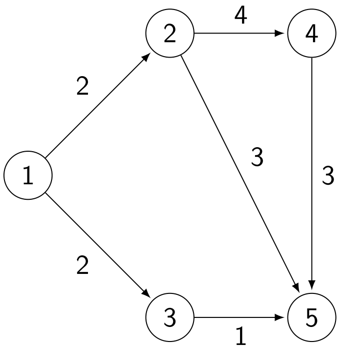
(a) A directed graph



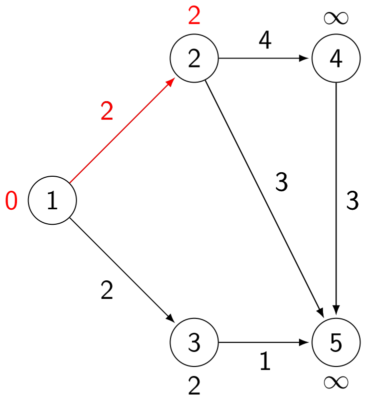
(b) Label-correcting

Arc $(1, 3)$ selected

Label-correcting algorithm: running example



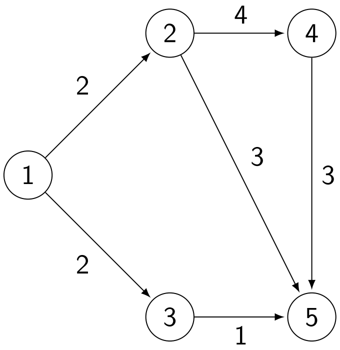
(a) A directed graph



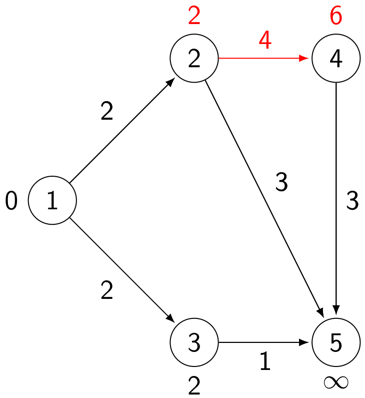
(b) Label-correcting

Arc $(1, 2)$ selected

Label-correcting algorithm: running example



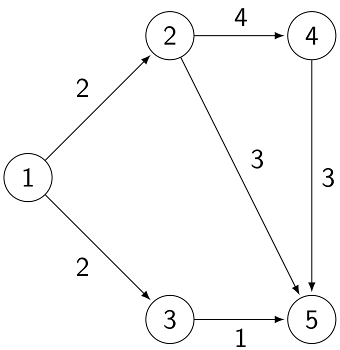
(a) A directed graph



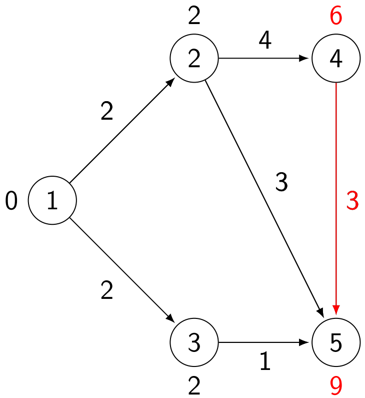
(b) Label-correcting

Arc $(2, 4)$ selected

Label-correcting algorithm: running example

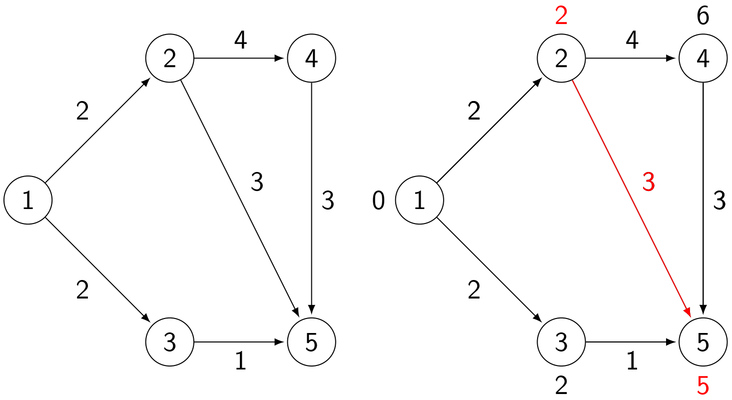


(a) A directed graph



(b) Label-correcting

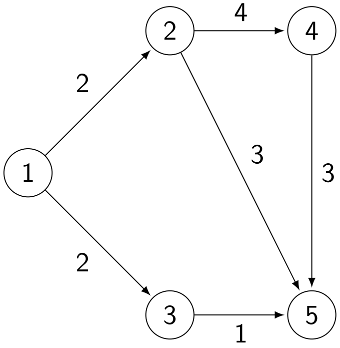
Arc $(4, 5)$ selected



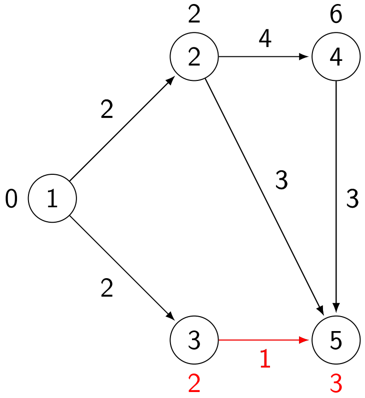
(a) A directed graph (b) Label-correcting

Arc $(2, 5)$ selected

Label-correcting algorithm: running example



(a) A directed graph



(b) Label-correcting

Arc $(3, 5)$ selected

Improved label-correcting algorithm

Algorithm Label-correcting

1: $d(s) \leftarrow 0$ and $\text{pred}(s) \leftarrow 0$
2: $d(j) \leftarrow \infty$ for each node $j \in N \setminus \{s\}$
3: **while** some arc (i, j) satisfies $d(j) > d(i) + c_{ij}$ **do**
4: $\quad d(j) \leftarrow d(i) + c_{ij}$
5: $\quad \text{pred}(j) \leftarrow i$
6: **end while**

- Searching for arcs that violate optimality conditions is time-consuming
- When to correct distance labels?
  1. First note labels of nodes can only decrease
  2. When node i's label decreases, we might have $d(j) > d(i) + c_{ij}$
  3. When a node's label decreases, add out-going edges to LIST for check

Label-correcting algorithm: finite-time termination

Algorithm Label-correcting

$d(s) \leftarrow 0$ and $\text{pred}(s) \leftarrow 0$
$d(j) \leftarrow \infty$ for each node $j \in N \setminus \{s\}$
while some arc (i, j) satisfies $d(j) > d(i) + c_{ij}$ **do**
    $d(j) \leftarrow d(i) + c_{ij}$
    $\text{pred}(j) \leftarrow i$
end while

Suppose the arc lengths are integers

- Each update decreases the distance label of some node by at least 1
- The range of distance label $[-(n-1)C, (n-1)C]$
- The total number of updates is bounded by $2n(n-1)C = O(n^2C)$

Label-correcting algorithm: finite-time termination

Algorithm Label-correcting

$d(s) \leftarrow 0$ and $\text{pred}(s) \leftarrow 0$
$d(j) \leftarrow \infty$ for each node $j \in N \setminus \{s\}$
while some arc (i, j) satisfies $d(j) > d(i) + c_{ij}$ **do**
    $d(j) \leftarrow d(i) + c_{ij}$
    $\text{pred}(j) \leftarrow i$
end while

Suppose the arc lengths are integers

- Each update decreases the distance label of some node by at least 1
- The range of distance label $[-(n-1)C, (n-1)C]$
- The total number of updates is bounded by $2n(n-1)C = O(n^2 C)$

This is an exponential-time algorithm!

What's bad? No good bound on the number of iterations.

Bellman-Ford algorithm: procedure

Algorithm Bellman-Ford

$d(s) \leftarrow 0$ and $\text{pred}(s) \leftarrow 0$
$d(j) \leftarrow \infty$ for each node $j \in N \setminus \{s\}$
for $k = 1 : n - 1$ **do**
  for $(i, j) \in A$ **do**
    if $d(j) > d(i) + c_{ij}$ **then**
      $d(j) \leftarrow d(i) + c_{ij}$
      $\text{pred}(j) \leftarrow i$
    end if
  end for
end for

- Clearly, the algorithm runs in $O(mn)$

Bellman-Ford algorithm: analysis

Algorithm Bellman-Ford

$d(s) \leftarrow 0$ and $\text{pred}(s) \leftarrow 0$
$d(j) \leftarrow \infty$ for each node $j \in N \setminus \{s\}$
for $k = 1 : n-1$ **do**
    for $(i, j) \in A$ **do**
        if $d(j) > d(i) + c_{ij}$ **then**
            $d(j) \leftarrow d(i) + c_{ij}$
            $\text{pred}(j) \leftarrow i$
        end if
    end for
end for

After k iterations

After k iterations, each distance label $d(i)$ is the length of the shortest $s - i$ path that uses k or fewer arcs provided such paths exist.

Thm: if no negative cycles, B-F finds shortest paths.

Detecting negative cycles: first attempt

Algorithm Bellman-Ford

$d(s) \leftarrow 0$ and pred$(s) \leftarrow 0$
$d(j) \leftarrow \infty$ for each node $j \in N \setminus \{s\}$
for $k = 1 : n - 1$ **do**
  for $(i, j) \in A$ **do**
    if $d(j) > d(i) + c_{ij}$ **then**
      $d(j) \leftarrow d(i) + c_{ij}$
      pred$(j) \leftarrow i$
    end if
  end for
end for

- When B-F algorithm terminates, there are two possibilities
  1. The distance labels do not satisfy the optimality condition
  2. The distance labels satisfy the optimality condition
- In case 1, negative cycles must exist (correctness of B-F)
- In case 2, negative cycles cannot exist

Detecting negative cycles: second attempt

- B-F does not identify a negative cycle
- B-F cannot terminate early
- B-F builds a *predecessor graph* G_{p} defined by $(\mathrm{pred}(i), i)$

Size of distance labels

If (i, j) is an arc in the predecessor graph, then $d(j) \geq d(i) + c_{ij}$.

Costs of paths

For an $h - \ell$ path in predecessor graph, the path cost at most $d(\ell) - d(h)$.

Appearance of cycles

The first cycle appearing in predecessor graph must have negative cost.

Detecting negative cycles: second attempt

Algorithm Bellman-Ford with cycle detection

$d(s) \leftarrow 0$ and $\text{pred}(s) \leftarrow 0$
$d(j) \leftarrow \infty$ for each node $j \in N \setminus \{s\}$
for $k = 1 : n - 1$ **do**
for $(i, j) \in A$ **do**
if $d(j) > d(i) + c_{ij}$ **then**
$d(j) \leftarrow d(i) + c_{ij}$
$\text{pred}(j) \leftarrow i$
if G_p contains a cycle $\mathcal{C}$ **then**
Return $\mathcal{C}$
end if
end if
end for
end for

- This algorithm rums in $O(mn^2)$ (DFS for detecting cycles in $O(n)$)
- Further improvement possible, running time can be reduced to $O(mn)$

All-pair shortest path problems

- How can we find shortest paths between every pair of nodes?
  - Naively run Dijkstra ($O(m \log_d n)$) or B-F ($O(mn)$) n times
- When there are negative arcs (no negative cycles)
  1. Run B-F once and transform to nonnegative arc case
     - For arc (i, j), set $c'_{ij} = d(i) + c_{ij} - d(j)$
  2. Run Dijkstra afterwards with c'
  3. Total running time $O(mn \log_d n)$

Why does it work?

All-pair shortest path problems

- How can we find shortest paths between every pair of nodes?
  - Naively run Dijkstra ($O(m \log_d n)$) or B-F ($O(mn)$) n times
- When there are negative arcs (no negative cycles)
  1. Run B-F once and transform to nonnegative arc case
     - For arc (i, j), set $c'_{ij} = d(i) + c_{ij} - d(j)$
  2. Run Dijkstra afterwards with c'
  3. Total running time $O(mn \log_d n)$

Why does it work?

Faster algorithm in $O(n^3)$ is available: Floyd-Warshall algorithm

Shortest path problems are not dead yet

2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)

Negative-Weight Single-Source Shortest Paths in Near-linear Time

Aaron Bernstein
Department of Computer Science
Rutgers University
New Brunswick, NJ, USA
bernstei@gmail.com

Danupon Nanongkai
MPI for Informatics,
University of Copenhagen,
and KTH
danupon@gmail.com

Christian Wulff-Nilsen
BARC, Department of Computer Science
University of Copenhagen
Copenhagen, Denmark
koolooz@di.ku.dk

***Abstract*—We present a randomized algorithm that computes single-source shortest paths (SSSP) in $O(m \log^8(n) \log W)$ time when edge weights are integral and can be negative.[1] This essentially resolves the classic negative-weight SSSP problem. The previous bounds are $\tilde{O}((m + n^{1.5}) \log W)$ [BLNPSSSW FOCS'20] and $m^{4/3+o(1)} \log W$ [AMV FOCS'20]. Near-linear time algorithms were known previously only for the special case of planar directed graphs [Fakcharoenphol and Rao FOCS'01].**

In contrast to all recent developments that rely on sophisticated continuous optimization methods and dynamic algorithms, our algorithm is simple: it requires only a simple graph decomposition and elementary combinatorial tools. In fact, ours is the first combinatorial algorithm for negative-weight SSSP to break through the classic $\tilde{O}(m\sqrt{n} \log W)$ bound from over three decades ago [Gabow and Tarjan SICOMP'89].

What's more, the new approach uses decades-old mathematical techniques, eschewing more sophisticated methods that have dominated modern graph theory research.

"I just couldn't believe such a simple algorithm exists," said Maximilian Probst Gutenberg, a computer scientist at the Swiss Federal Institute of Technology Zurich. "All of it has been there for 40 years. It just took someone to be really clever and determined to make it all work."

Upcoming

Week 1-8 (AU4606 & AI4702):

- Introduction (1 lecture)
- Preparations (3 lectures)
  - basics of graph theory
  - algorithm complexity and data structure
  - graph search algorithm
- Shortest path problems (this & next lecture)
- Maximum flow problems (5 lectures)
- Minimum cost flow problems (3 lectures)
- Introduction to multi-agent systems (1 lecture)
- Introduction to cloud networks (1 lecture)

Week 9-16 (AU4606):

- Simplex and network simplex methods (2 lectures)
- Global minimum cut problems (3 lectures)
- Minimum spanning tree problems (3 lectures)