IE 604: Network Flow Optimization[1]

Course Notes, Part II: Shortest Paths

Dr. Jim Ostrowski

University of Tennessee - Knoxville

[1]Last updated: February 9, 2022

Section 1

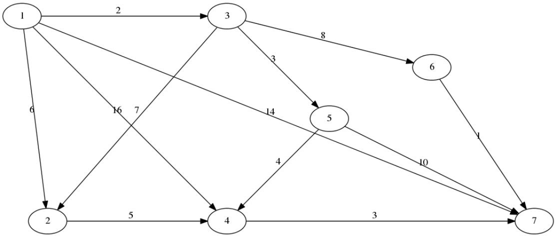
Introduction

Unit 1

The Shortest Path Problem

Notation

- Directed network, $G = (N, A)$
- Arcs, $(i, j) \in A$
    - length (cost): c_{ij}
    - max length: $C = \max_{ij}\{c_{ij}\}$
- Adjacency list of node i, A_i
- Source, s



Problem Definition

- Length perspective
  - for every nonsource node $i \in N$, find a shortest length directed path from node s to node i
    - the length of a path is the sum of the lengths of the arcs on the path
- Minimum cost flow perspective
  - find a way to send 1 unit of flow as cheaply as possible from node s to every other node $i \in N \setminus \{s\}$ in an uncapacitated network

Linear Programming Formulation

$$
\begin{aligned}
\text{Minimize} \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = \begin{cases} n-1 & \text{for } i = s \\ -1 & \forall i \in N \setminus \{s\} \end{cases} \\
& x_{ij} \geq 0.
\end{aligned}
$$

Linear Programming Example

How do we obtain the shortest paths from the optimal solution?

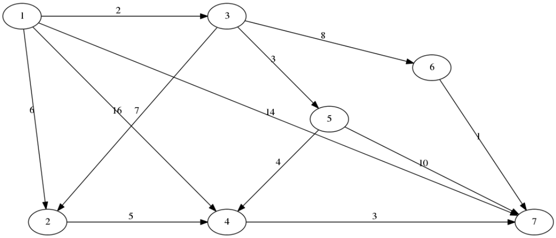
LP: single destination

Linear Programming Dual: single destination

Linear Programming Dual: single destination (cont.)

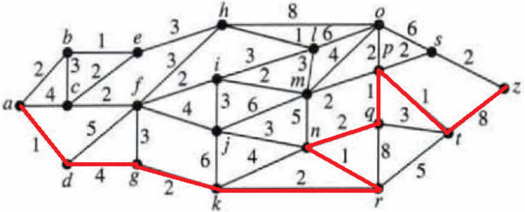
Assumptions

1. All arc lengths are integers
2. The network contains a directed path from node s to every other node in the network
3. The network does not contain a negative cycle
4. Network is directed



Main Types of Shortest Path Problems

1. Single-source acyclic graph
2. Single-source with non-negative arc lengths on general graph
3. Single-source with arbitrary arc lengths on acyclic graph
4. Single-source with arbitrary arc lengths on general graph
5. All-pairs shortest path problem



Other Problems Related to the Shortest Path Problem

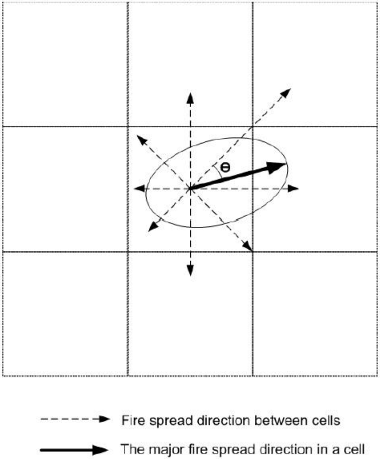
1. Maximum capacity path problem
2. Maximum reliability path problem
3. Shortest path problem with turn penalties
4. Shortest path with an additional constraint
5. Resource-constrained shortest path problem

Unit 2

Applications

Modeling the Behavior of a Wildfire

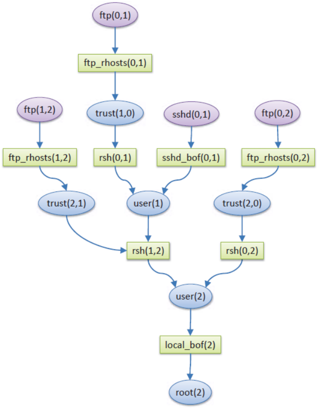
- A landscape can be modeled as a grid graph
  - nodes represent center points of cells
  - the weights on arcs represent the travel time between two cell midpoints (computed using ellipse model)
    - $R = \frac{b^2 - c^2}{b - c \times \cos(\theta)}$, $(0 \leq \theta < \pi/2)$
    - $R = \frac{b^2 - c^2}{b - c \times \cos(\pi - \theta)}$, $(\pi/2 \leq \theta \leq \pi)$
- The time needed for a fire to spread from cell s to cell t is the shortest path



Modeling the Behavior of a Wildfire (2)

Modeling the Behavior of a Computer Hacker

- Computer security researchers have developed a tool called an attack graph to visualize the vulnerability of a computer network
  - nodes represent the privilege / identity that the attacker has obtained
  - arcs represent exploits that can be use to escalate the privilege / identity
- We could assign weights to the edges in the attack graph to represent the time needed to
- The minimum time/cost for the hacker to gain `root` access is the shortest path from a leaf node to the root node



Section 2

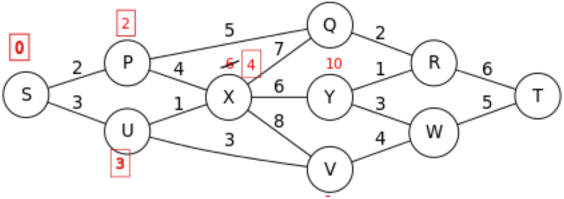
Introduction to Shortest Path Algorithms

Unit 1

Types of Shortest Path Algorithms

Label Algorithms

- Assign tentative distance labels to nodes at each step
  - labels represent upper bounds on the shortest path to each node



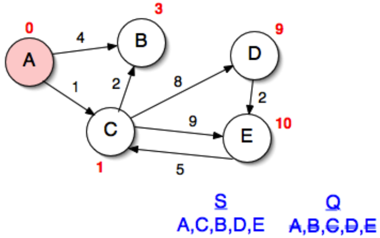
Label-Setting Algorithms

- Procedure
  - designate one label as permanent in each iteration
- Advantages
  - more efficient
- Disadvantages
  - only works for graphs without a negative cycle

Label-Setting Algorithms

Label-Correcting Algorithms

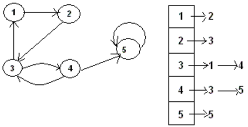
- Procedure
  - consider all labels as temporary until the last step in which they all become permanent
- Advantages
  - work for all classes of problems
  - offer more algorithmic flexibility
- Disadvantages
  - less efficient



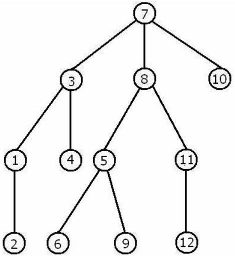
Unit 2

How to Store the Shortest Paths

Shortest Path Tree



- Directed-out tree
  - rooted at node s
- Unique path from s to a node i in the tree is a shortest path from s to i
- Why is this true?



Property 4.1

- If the path $s = i_1 - i_2 - \cdots - i_h = k$ is a shortest path from node s to node k, then for every $q = 2, 3, \ldots, h-1$, the subpath $s = i_1 - i_2 - \cdots - i_q$ is a shortest path from the source node to node i_q

Proof.

(By contradiction)

Property 4.2

Let $d(i)$ be the shortest path from s to i

- A directed path P from s to k is a shortest path if and only if $d(j) = d(i) + c_{ij}$ for all $(i, j) \in P$

Proof.

Property 4.2 (Cont.)

Why Does the Tree Contain the Shortest Paths?

1. The network contains a finite number of paths; thus, there is a shortest path to every node
2. Then, by Property 4.2, we can always find a shortest path from s to an node such that

$$d(j) = d(i) + c_{ij} \, \forall (i, j) \in P \tag{1}$$

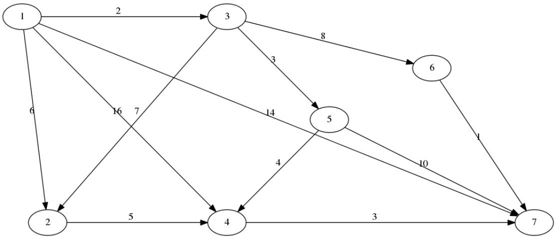
3. Perform breadth-first search on the network using the arcs that satisfy (1), creating a breadth-first tree

Unit 3

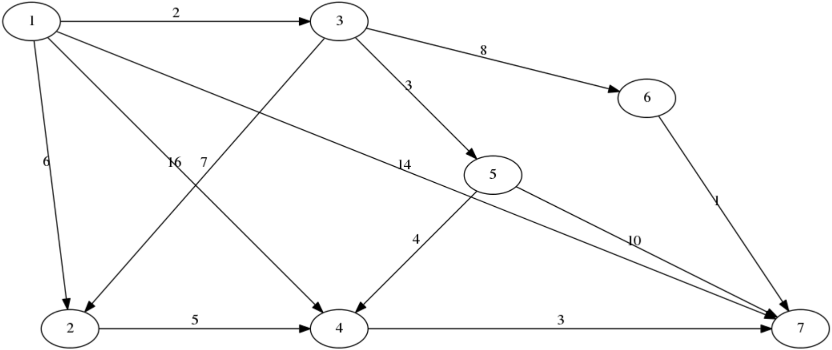
Acyclic Networks

Shortest Path on Acyclic Networks

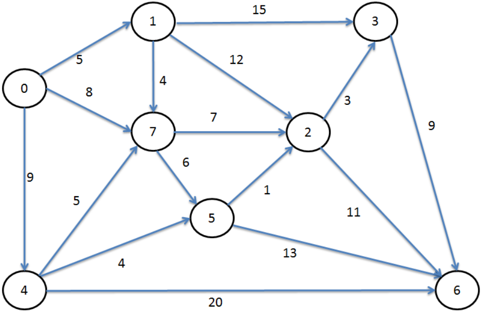
- A simple "sweeping" algorithm is optimal
  - number nodes according to their topological ordering
  - set distance label $d(s) = 0$ and the others to a very large number
- for i in *TopologicallyOrderedSet*
  - for $(j, i) \in A_i^-$
    - if $d(i) > d(j) + c_{ji}$ then $d(i) \leftarrow d(j) + c_{ji}$



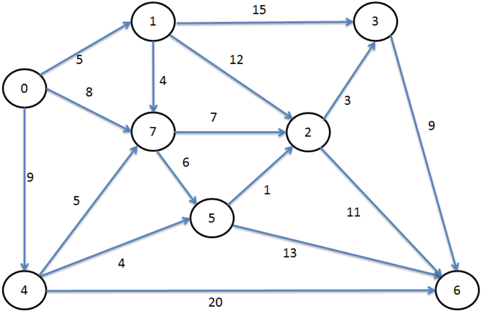
Example



Practice



Solution



Why Does It Work?

Proof.

(By induction.)

Why Does It Work? (2)

Worst Case Complexity

- Each arc is “examined” exactly once
  - addition
  - comparison
  - possible assignment
- Thus, the algorithm solves the problem in $O(m)$ time

Section 3

Dijkstra's Algorithm

Unit 1

Introduction

About Dijkstra's Algorithm

- Edsger Dijkstra (1930-2002)
  - conceived his algorithm in 1956
  - published it in 1959
- Label-setting algorithm
- Single-source shortest path problem
- Differences with algorithm for acyclic networks
  - can be used on any graph without negative cycles
  - may examine an arc more than once

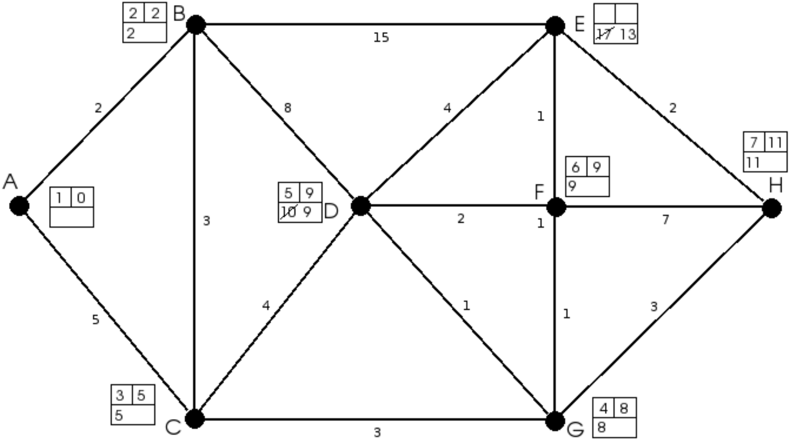
Unit 2

The Algorithm

Terminology

- Nodes divided into two groups
  - permanently labeled, S
  - temporarily labeled, $\bar{S}$
- Label values, $d(i)$
  - distance to a permanently labeled node is the shortest distance to that node
  - distance to temporarily labeled node is an upper bound on the shortest distance to that node
  - all labels are the distances along paths whose internal nodes are permanently labeled

Terminology (2)



Algorithm Summary

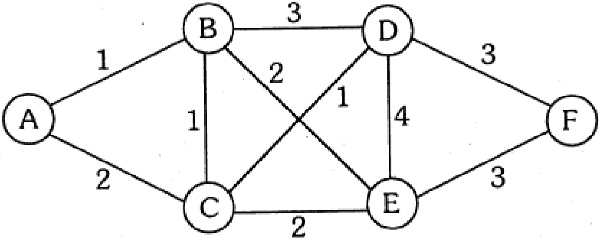
- Basic idea
  - start at node s
  - while there remain temporary nodes
    - **node selection:** choose neighbor with smallest temporary label; make this node a permanent node
    - update the temporary labels of all the neighbors of the new permanent node

$$d(i) = \min_{(j,i)\in A}\{d(j) + c_{ji}\}$$

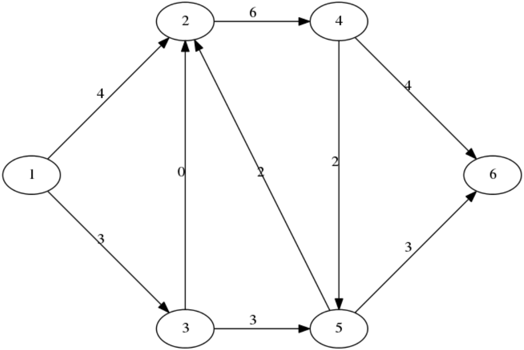
Unit 3

Examples

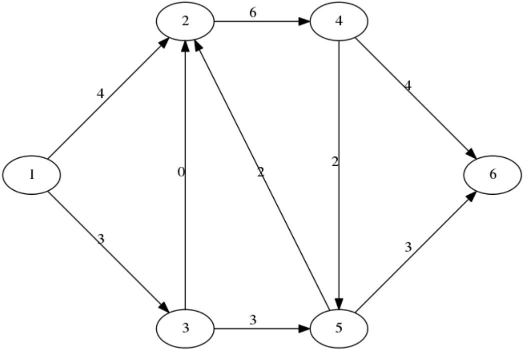
Example



Practice



Solution



Unit 4

Technical Details

Pseudocode

- Initialize
  - $S := \emptyset; \bar{S} = N$
  - $d(s) = 0$ and $pred(s) = 0$
  - $d(i) := \infty$ for all $i \in N \setminus \{s\}$
- **while** $|S| < n$
  - let $i \in \bar{S}$ be a node for which $d(i) = \min\{d(j) : j \in \bar{S}\}$
  - $S := S \cup \{i\}; \bar{S} := \bar{S} \setminus \{i\}$
  - **for** each $(i, j) \in A_i^+$
    - **if** $d(j) > d(i) + c_{ij}$ **then** $d(j) := d(i) + c_{ij}$ and $pred(j) := i$

- Proof by mathematical induction
- Induction hypotheses
  1. the distance label of each node in S is optimal
  2. the distance label of each node in S is the shortest path length from the source provided that each internal node in the path lies in S

Correctness

Hypothesis #1

Correctness

Hypothesis #1 (2)

Correctness

Hypothesis #2

Worst-Case Complexity

- **Node selection**
  - performed n times
  - each time requires scanning all temporary nodes (up to n nodes)
  - $n + (n-1) + (n-2) + \cdots + 1 = O(n^2)$
- **Distance updates**
  - performed $|A_i|$ times for each node
  - Overall, $\sum_{i \in N} |A_i| = m$ times
  - each time requires $O(1)$ time
  - $O(m)$ total time
- **Overall**
  - $O(n^2 + m) = O(n^2)$
  - cannot get better for completely dense graphs

Section 4

Introduction to Label-Correcting Algorithms

Unit 1

Introduction

Motivation

- Label setting algorithms only work for acyclic networks or networks with non-negative costs (i.e., networks without negative cycles)
- In general, we cannot easily solve networks with negative cycles
- We would be willing to accept an algorithm that can do the following:
  - identifies a negative cycle when one exists
  - solves the problem if no negative cycle exists

How They Work

- Generic algorithm
  - reduces the distance label of one node at each iteration by considering only local information
  - all labels are made permanent when the algorithm terminates
- More advanced algorithms use optimality conditions

- **If** the distance labels $d(j)$ are shortest path distances, **then** they must satisfy the following necessary optimality conditions

$$d(j) \leq d(i) + c_{ij} \quad \forall (i,j) \in A \tag{2}$$

- Proof
  - if this inequality is violated for any arc, then we can reduce the shortest path (contradicting the optimality of the distance labels) to node j by setting it to

$$d(j) = d(i) + c_{ij}$$

- **If** each $d(j)$ represents the length of some directed path from s to node j and this solution satisfies the conditions (2), **then** it must be optimal (i.e., the shortest path)
- Proof

Optimality Conditions: Proof (Cont.)

Theorem

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

$$d(j) \leq d(i) + c_{ij} \quad \forall (i,j) \in A$$

Properties About Reduced Arc Lengths

- Define the reduced arc length of an arc (i, j) with respect to the distance labels $d(\cdot)$, as
  - $c_{ij}^d = c_{ij} + d(i) - d(j)$
- Properties
  - For any directed cycle W
    - $\sum_{(i,j)\in W} c_{ij}^d = \sum_{(i,j)\in W} c_{ij}$
  - For any directed path P from node k to node ℓ
    - $\sum_{(i,j)\in P} c_{ij}^d = \sum_{(i,j)\in P} c_{ij} + d(k) - d(\ell)$
  - If $d(\cdot)$ represent shortest path distances, $c_{ij}^d \geq 0$ for all $(i, j) \in A$

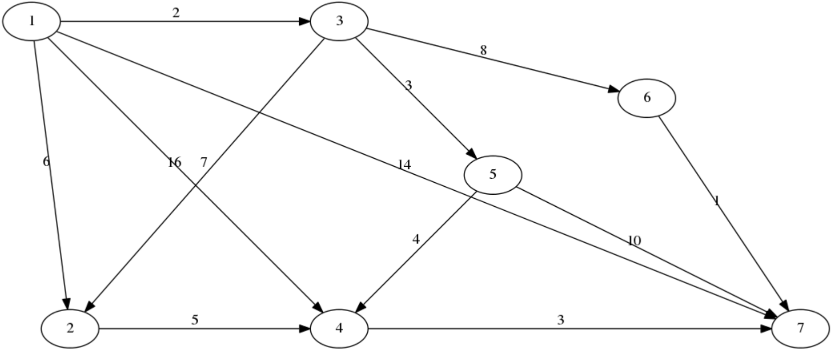
Unit 2

The Generic Label-Correcting Algorithm

The Generic Label-Correcting Algorithm

- Assumptions
  - Assumes a graph with no negative cycles
- Procedure
  - Find some arc (i, j) that violate optimality conditions
    - $d(j) \leq d(i) + c_{ij}$
  - Update the distance label for the arc head
    - $d(j) \leftarrow d(i) + c_{ij}$
  - Terminate when all arcs satisfy the optimality conditions

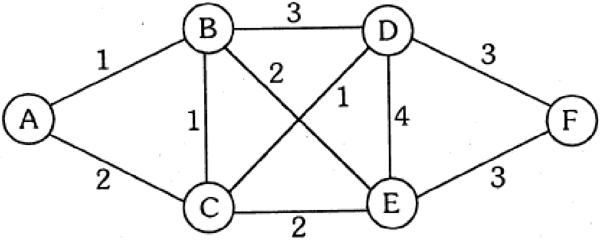
The Generic Label-Correcting Algorithm



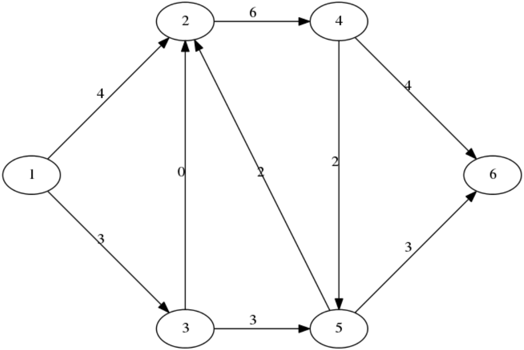
Predecessor Graph

- Directed out-tree T rooted at s
  - collection of arcs $(pred(j), j)$
- When a distance update is made, one arc is removed and another added

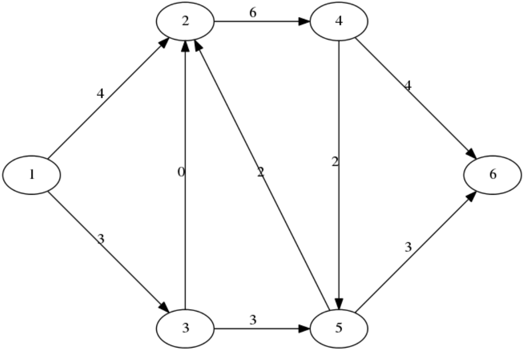
Example



Practice



Solution



Pseudocode

- Initialize
  - $d(s) := 0$ and $pred(s) := 0$
  - $d(j) := \infty$for all $j \in N \setminus \{s\}$
- **while** some arc (i, j) satisfies $d(j) > d(i) + c_{ij}$
  - $d(j) := d(i) + c_{ij}$
  - $pred(j) := i$

Worst-Case Complexity

- Assume data are integral
- Let C be the maximum absolute value of arc costs
- For each finite $d(j)$, $-nC \leq d(j) \leq nC$ because each path contains at most $n-1$ arcs
- Algorithm updates any label $d(j)$ at most $2nC$ times
- Algorithm performs $O(n^2C)$ iterations

Unit 3

Modified Label-Correcting Algorithm

- How do we find an arc that violates the optimality condition?
  - scan the arc list
    - how much time does this require?
  - can we do better than a scan?

List of Potentially Violating Arcs

- Maintain a list, $LIST$, of all arcs that *might* violate their optimality conditions
- Steps
  1. Select some arc $(i, j) \in LIST$ that violates optimality conditions
     1. $d(j) > d(i) + c_{ij}$
  2. Remove (i, j) from $LIST$
  3. Update the distance label $d(j) = d(i) + c_{ij}$
     1. this may cause some arcs in A_j to violate their optimality condition
     2. decreasing $d(j)$ maintains the optimality condition of for all incoming arcs at node j
  4. Add arcs in A_j to $LIST$

Modified Label-Correcting Algorithm

How it Works

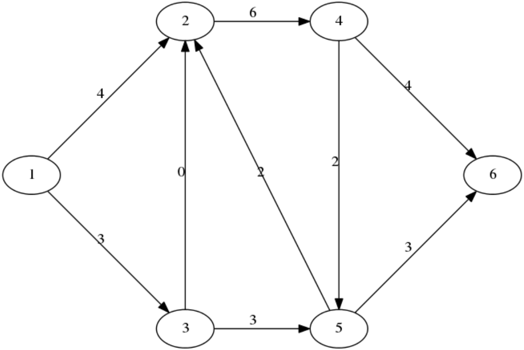
- Instead of holding arcs in $LIST$, hold nodes with this property:
  - if an arc (i, j) violates the optimality condition, $LIST$ must contain node i

Modified Label-Correcting Algorithm

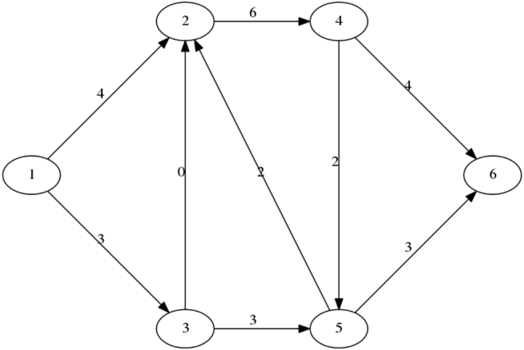
Pseudocode

- Maintain a list, $LIST$, of all nodes that *might* violate their optimality conditions
- **while** $LIST \neq \emptyset$
  1. remove some node $i \in LIST$
  2. **for** each $(i, j) \in A_i$
     if $d(j) > d(i) + c_{ij}$
     $d(j) = d(i) + c_{ij}$
     $pred(j) = i$
     if $j \neq LIST$ **then** add j to $LIST$

Practice



Solution



Worst-Case Complexity

- Each update of $d(j)$
  - scan each arc in A_i^+: $O(|A_i^+|)$
- Algorithm can update a label at most $O(2nC)$ times
- Overall bound
  - $\sum_{i \in N} (2nC)|A_i^+| = O(nmC)$

Section 5

Special Implementations of the Modified Label-Correcting Algorithm

Unit 1

Introduction

Why Modify It?

- The generic algorithm does not run in polynomial time
  - $O(n^2 C)$

Unit 2

$O(nm)$ Implementation

- For $n - 1$ iterations:
  - for all $(i, j) \in A$:
    - if $d(j) > d(i) + c_{ij}$, then set $d(j) := d(i) + c_{ij}$

Correctness and Complexity

Theorem

The label-correcting algorithm requires $O(nm)$ time as long as we examine all arcs at every pass

Proof.

(By induction)

□

Correctness and Complexity: Proof (Cont.)

FIFO Version

- We don't need to consider every $(i, j) \in A$ during every pass
  - if a node's distance label doesn't change during a pass, we can ignore it during the next pass
- Implementation
  - if a node's distance label changes during a pass, add it to a FIFO queue

Unit 3

Detection of Negative Cycles

Modification to Label-Correcting Algorithm

- Recall that:
  - $-nC \leq d(j) \leq nC$
- Thus, if a distance label falls below $-nC$, then there exists a negative cycle

Modification #2 ($\mathcal{O}(n)$)

DetectCycleInPredecessorGraph
Label source node.
some node is unlabeled
choose unlabeled node k
label k
trace predecessor indices starting at node k, assigning label k to all nodes encountered until first unlabeled node ℓ is reached
nodes k and ℓ have the same labelsreturn certificate that the PG contains a negative cycle
return certificate that PG has no negative cycle

- if PG contains a cycle, then G contains a negative cycle
- apply check after every αn distance label updates (does not add to worst-case complexity)

Section 6

All-Pairs Shortest Paths

Unit 1

Introduction

All Pairs Shortest Path Problem

- Problem statement
  - "Find the shortest paths between all pairs of nodes in a network"
- Assumptions
  - network is strongly connected
  - no negative cycles
- Algorithms
  1. Repeated shortest path algorithm
  2. All-pairs label-correcting algorithm
     1. generic version
     2. Floyd-Warshall implementation

Unit 2

Repeated Shortest Path Algorithm

How it Works

- If the network contains negative arcs
    - use the FIFO label-correcting algorithm to transform network
- Solve single-source shortest path algorithm n times on the transformed network (each time using a different node as a source)

Transforming Network With Negative Arcs

FIFO Label-Correcting Algorithm ($\mathcal{O}(nm)$)

- Procedure
  - **If** the network contains a negative cycle, the algorithm will detect it
  - **Else**
    - compute shortest distances $d(j)$ from a source s to all other nodes ($\mathcal{O}(nm)$)
    - set the cost of each arc as the reduced cost: $c_{ij}^d := c_{ij} + d(i) - d(j)$
- Computing the shortest path distances in the original network
  - let $\sum_{(i,j)\in P_{\ell k}} c_{ij}$ be the shortest path length from k to ℓ on the transformed network
  - the shortest path length on the original network is $\sum_{(i,j)\in P_{\ell k}} c_{ij} + d(\ell) - d(k)$ [see property 5.2(c)]

Complexity

- Let $S(n, m, C)$ be the time needed to solve a shortest path problem with non-negative arc lengths
- The FIFO label-correcting algorithm takes $\mathcal{O}(nm)$ time
- Overall complexity
    - $\mathcal{O}(nm + nS(n, m, C)) = \mathcal{O}(nS(n, m, C))$

Unit 3

All Pairs Optimality Conditions

Notation

- Distance label, $d[i, j]$
  - represents the length of some directed walk from node i to node j
  - is an upper bound on the shortest path from i to j
  - if no walk exists, is infinite

Theorem

Theorem

(All-Pairs Shortest Path Optimality Conditions). For every pair of nodes $[i, j] \in N \times N$*, let* $d[i, j]$ *represent the length of some directed path from node* i *to node* j *satisfying* $d[i, i] = 0$ *for all* $i \in N$ *and* $d[i, j] \leq c_{ij}$ *for all* $(i, j) \in A$*. These distances represent all-pairs shortest path distances if and only if they satisfy the following all-pairs shortest path optimality conditions:*

$$d[i, j] \leq d[i, k] + d[k, j] \text{ for all nodes } i, j, \text{and } k$$

Unit 4

All-Pairs Generic Label-Correcting Algorithm

Summary

- Start with some distance labels $d[i, j]$
- Successively update these until they satisfy the optimality conditions

Pseudocode

AllPairsGenericLabelCorrecting
$d[i,j] := \infty$ for all $[i,j] \in N \times N$
$d[i,i] := 0$ for all $i \in N$
$d[i,j] := c_{ij}$ for all $(i,j) \in A$
$d[i,j] > d[i,k] + d[k,j]$ for some nodes i, j, k
$d[i,j] > d[i,k] + d[k,j] d[i,j] := d[i,k] + d[k,j]$

Complexity

- Assume data are integral
- Let C be the maximum absolute value of arc costs
- For each finite $d[i,j]$,$-nC \leq d[i,j] \leq nC$ because each path contains at most $n-1$ arcs
- Algorithm updates any label $d[i,j]$ at most $2nC$ times
- The graph has n^2 pairs of nodes
- Algorithm performs $O(n^3C)$ iterations

Unit 5

Floyd-Warshall Algorithm

Key Property

- Let $d^k[i,j]$ represent the length of the shortest path from i to j subject to the condition that the path only uses nodes $1, 2, \ldots, k-1$ as internal nodes
- Property

$$d^{k+1}[i,j] = \min\left\{d^k[i,j],\ d^k[i,k] + d^k[k,j]\right\}$$

1. Compute $d^1[i,j]$ for all $i,j \in N \times N$
2. Use key property to compute $d^2[i,j]$ given $d^1[i,j]$ for all $i,j \in N \times N$

$$d^2[i,j] = \min\left\{d^1[i,j],\, d^1[i,k] + d^1[k,j]\right\}$$

3. Continue until $d^{n+1}[i,j]$ is computed for all $i,j \in N \times N$

Algorithm maintains predecessor indices, $pred[i,j]$, which denote the last node prior to j in the shortest path from i to j

Pseudocode

FloydWarshall

$d[i,j] := \infty$ and $pred[i,j] := 0$ for all $[i,j] \in N \times N$

$d[i,i] := 0$ for all $i \in N$

$d[i,j] := c_{ij}$ and $pred[i,j] := i$ for all $(i,j) \in A$

1 to n

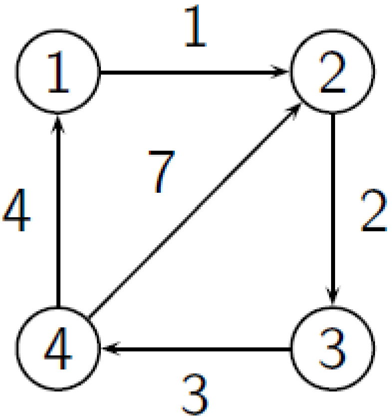
$[i,j] \in N \times N d[i,j] > d[i,k] + d[k,j] d[i,j] := d[i,k] + d[k,j]$

$pred[i,j] := pred[k,j]$

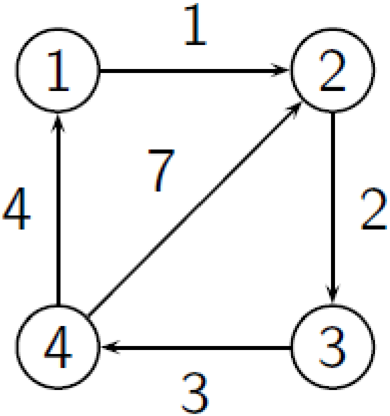
Complexity

- Major iterations: n (one for each k)
- Computations per iteration
  - $\mathcal{O}(1)$ for each node pair
- Overall complexity
  - $\mathcal{O}(n \times n^2) = \mathcal{O}(n^3)$

Example



Example: Initialization



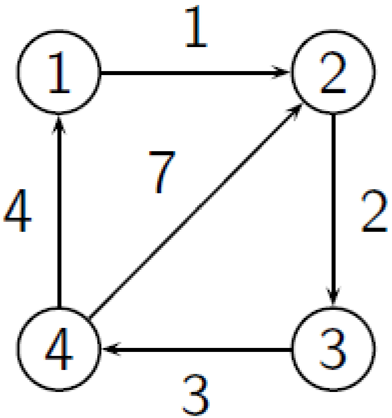
Distances

	1	2	3	4
1				
2				
3				
4				

Predecessors

	1	2	3	4
1				
2				
3				
4				

Example: 1



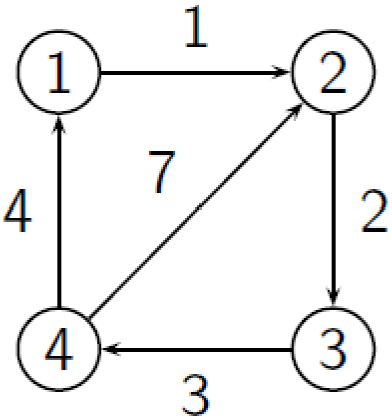
Distances

	1	2	3	4
1				
2				
3				
4				

Predecessors

	1	*2*	*3*	*4*
1				
2				
3				
4				

Example: 2



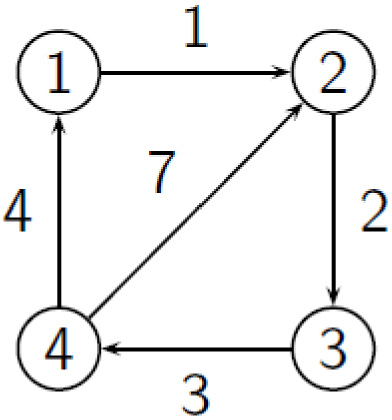
Distances

	1	2	3	4
1				
2				
3				
4				

Predecessors

	1	*2*	*3*	*4*
1				
2				
3				
4				

Example: 3



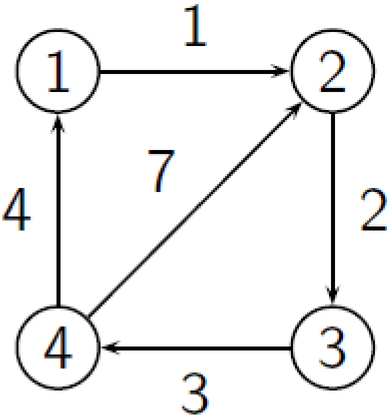
Distances

	1	2	3	4
1				
2				
3				
4				

Predecessors

	1	*2*	*3*	*4*
1				
2				
3				
4				

Example: 4



Distances

	1	2	3	4
1				
2				
3				
4				

Predecessors

	1	*2*	*3*	*4*
1				
2				
3				
4				

Unit 6

Detection of Negative Cycles

Generic All-Pairs Label-Correcting Algorithm

1. If $i = j$, check whether $d[i, i] < 0$
2. If $i \neq j$, check whether $d[i, j] < -nC$

Floyd-Warshall Algorithm

1. Check if $d[i,i] < 0$ whenever updating $d[i,i]$ for some node i
2. Check if predecessor graph contains a negative cycle