Logistics systems planning I
Optimization of logistics systems
Transportation planning – Shortest path problems (Part 2)

Univ.-Prof. Dr. Michael Schneider

Deutsche Post Chair – Optimization of Distribution Networks (DPO)
RWTH Aachen University

schneider@dpo.rwth-aachen.de

Course agenda

1. Fundamentals
2. Transportation planning
   - Introduction
   - **Shortest path problems**
   - Minimum spanning tree problem
   - Traveling salesman problem
   - Vehicle routing problems
   - Arc routing problems
3. Warehouse planning
4. Introduction to location planning

Agenda

1. Shortest path problems
   - Dijkstra algorithm
   - FIFO algorithm
   - Floyd-Warshall algorithm

Dijkstra algorithm

- Most well-known shortest path algorithm (Dijkstra 1959, Dantzig 1960)
- Assumes non-negative arc weights c_{ij}
- In each iteration:
  - select node $i \in V$
  - examine outgoing arcs $(i, j) \in \delta^+(i)$ and associated labels $d(j)$
- Distinction between unmarked, temporarily marked, and ultimately marked nodes

Dijkstra algorithm – ideas

- Source node s temporarily marked with $d(s) = 0$, all other unmarked
- In each iteration, select temporarily marked node i with minimum label value $d(i)$
- Successor nodes $j : (i, j) \in \delta^+(i)$ are temporarily marked with $d(j) := d(i) + c_{ij}$ if label improves. Afterwards, node i is ultimately marked
- Selection of node i with minimum label guarantees that no shorter path to this node can exist
- Non negative arc weights $\rightarrow$ nodes i are traversed in non-decreasing order of label values

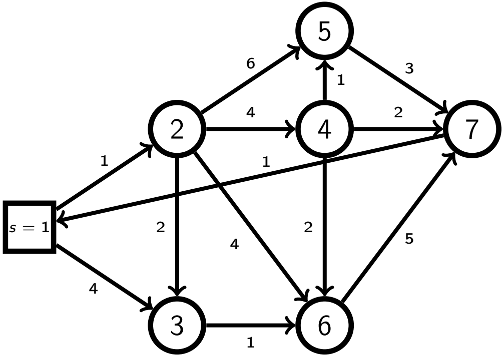
Dijkstra algorithm – pseudocode

Algorithm 1: Dijkstra algorithm

```
// Initialization
SET d(s) := 0, pred(s) := undef, L := {s}
SET d(j) := ∞ for all j ∈ V \ {s}
// Loop
while L ≠ ∅ do
    SELECT i ∈ L with d(i) = min_{k∈L} d(k)
    SET L := L \ {i}  // set i as ultimately marked
    for j : (i,j) ∈ δ+(i) do
        if d(j) > d(i) + c_ij then
            // Update label
            SET d(j) := d(i) + c_ij; pred(j) := i
            SET L := L ∪ {j}
// Output:  predecessors pred(·) and distances d(·)
```

Set L contains all temporarily marked nodes. Note: each node i is selected exactly once.

Dijkstra algorithm – example



$(\cdot)$ temporarily marked, $[\cdot]$ ultimately marked

Dijkstra algorithm – example

Iteration	selected node i	temporarily marked	new labels
Init.	–	$s = 1$	d(1)=0
1	1	2,3	d(2)=1, d(3)=4
2	2	3,4,5,6	d(3)=3, d(6)=d(4)=5, d(5)=7
3	3	4,5,6	d(6)=4
4	6	4,5,7	d(7)=9
5	4	5,7	d(5)=6, d(7)=7
6	5	7	–
7	7	–	–

Result

node i	1	2	3	4	5	6	7
distance $d(i)$	0	1	3	5	6	4	7
predecessor $pred(i)$	*undef*	1	2	2	4	3	4

Comments

- When looking for a shortest path from s to t, the algorithm can be stopped as soon as t is ultimately marked
- In undirected graphs (edges instead of arcs), replace each edge by two anti-parallel arcs

- Runtime mostly determined by the selection of nodes with minimal labels (node i with $d(i) = \min_{k \in L} d(k)$).
  Efficient implementations differ with regards to the data structures used to store set L of temporarily marked nodes (see Ahuja et al. 1993)

Dijkstra algorithm – visualizations

- `https://qiao.github.io/PathFinding.js/visual/`
- `https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html`
- `https://www-m9.ma.tum.de/graph-algorithms/spp-dijkstra/index_de.html`

Agenda

FIFO algorithm – prerequisites

- The following label-correcting algorithms
  - require that no cycles with negative length exist
  - allow arc weights $c_{ij} < 0$ for some arcs (i, j)
  - allow directed cycles
- FIFO algorithm (s-to-all shortest path problem)
- Floyd-Warshall algorithm (all-pairs shortest path problem)

FIFO algorithm – ideas I

- Order of arc selection strongly affects the runtime of our generic shortest path algorithm
- Naive implementation:
  - arbitrary order of arcs $(i, j) \in A$: Check optimality condition in this order and update labels of nodes if necessary.
  - repeat until optimality condition is met: $d(j) \leq d(i) + c_{ij}, \forall (i, j) \in A$.
- All arcs are traversed at most $(|V| - 1)$ times
  - all node labels having a shortest path which consists of k arcs are correct after kth iteration (proof by induction over k).
  - worst-case complexity of complete procedure is $\mathcal{O}(|V||A|)$.

FIFO algorithm – ideas II

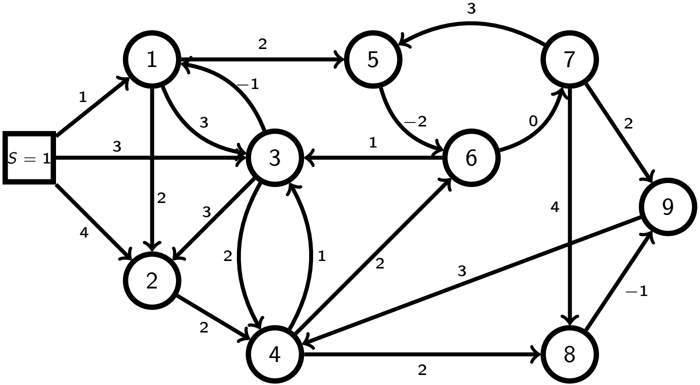
- FIFO builds upon this idea but considers that label $d(j)$ can only change if label $d(i)$ has changed for an arc $(i, j) \in A$
- If $d(j)$ changes, all arcs $(j, k) \in \delta^+(j)$ must be examined in next iteration
- Instead of storing the information about these arcs, it is sufficient to store the nodes j themselves
- Store nodes j in a queue following the FIFO principle (first in – first out).

FIFO algorithm – pseudocode

Algorithm 2: FIFO algorithm

```
// Initialization
SET d(s) := 0, pred(s) := undef, Q := (s)
SET d(j) := ∞ for all j ∈ V \ {s}
// Loop
while Q ≠ ∅ do
    REMOVE first element i ∈ Q from queue
    for j : (i,j) ∈ δ⁺(i) do
        if d(j) > d(i) + c_ij then
            SET d(j) := d(i) + c_ij
            SET pred(j) := i
            if j ∉ Q then
                APPEND j to queue Q
// Output:  predecessors pred(·) and distances d(·)
```

FIFO algorithm – example I

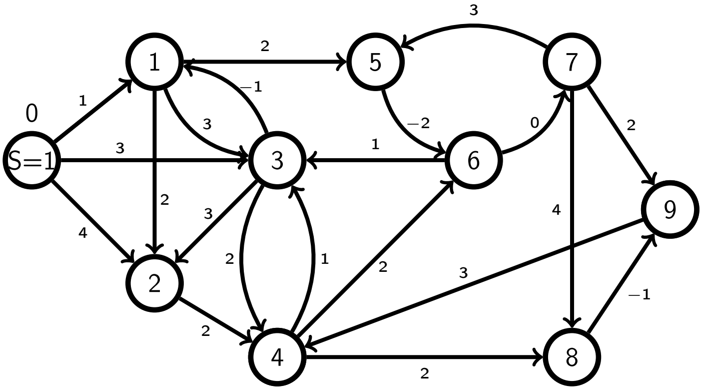


Graph contains cycles and arcs with negative weights. Examples: $C_1 = (1, 3, 1)$, $C_2 = (5, 6, 7, 5)$, or $C_3 = (1, 5, 6, 7, 9, 4, 3, 1)$. No cycle with negative length

FIFO algorithm – example II

Iteration	selected node i	Queue Q after iteration	new labels $d(j) < \infty$
Init.	—	s	$d(s) = 0$
1	s		
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			

FIFO algorithm – example III



Queue Q: s

Agenda

1. Shortest path problems
   - Dijkstra algorithm
   - FIFO algorithm
   - Floyd-Warshall algorithm

Floyd-Warshall algorithm

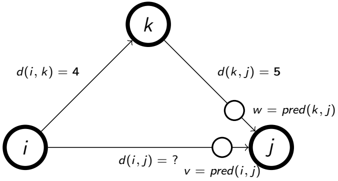
- Solution of shortest path problem for all pairs of nodes
  - solve *s*-to-all shortest path problem using each node as source node *s*
  - depending on characteristics of digraph (cycles, positive arc weights): apply previous algorithms
- Simultaneous approach: Floyd-Warshall algorithm (also known as triple algorithm)
- only prerequisite: no cycles with negative length

Floyd-Warshall algorithm – ideas I

- Introduction of labels
  - $d(i,j)$: upper bound for length of path from i to j.
  - $pred(i,j)$: predecessor of j on path from i to j.
- Initialization of labels
  - $d(i,i) := 0$, $pred(i,i) := undef$ for all nodes $i \in V$ (path from i to i has length 0)
  - $d(i,j) := c_{ij}$, $pred(i,j) := i$ for all arcs $(i,j) \in A$ (arc (i,j) is path from i to j, not necessarily minimal)
  - $d(i,j) := \infty$ for all remaining pairs $(i,j) \notin A$, $i \neq j$ (no path from i to j is known)

Shortest Path

- Label for path from i to j is updated if $d(i,j) > d(i,k) + d(k,j)$:
  - $d(i,j) := d(i,k) + d(k,j)$
  - $pred(i,j) := pred(k,j)$



Before update:

- $d(i,j) := 10$
- $pred(i,j) := v$

After update:

- $d(i,j) := d(i,k) + d(k,j) = 9$
- $pred(i,j) := w$

Optimality conditions

Optimality conditions

Given a weighted digraph $D = (V, A, c_{ij})$ and a set of labels $d(i, j)$ with $i, j \in V$. It holds:

For each pair of nodes $(v, w) \in V \times V$, $d(v, w)$ is the length of a shortest path from v to w

if and only if

the labels satisfy the optimality conditions

$$d(i, j) \leq d(i, k) + d(k, j) \qquad \forall (i, j, k) \in V \times V \times V.$$

Stopping criterion

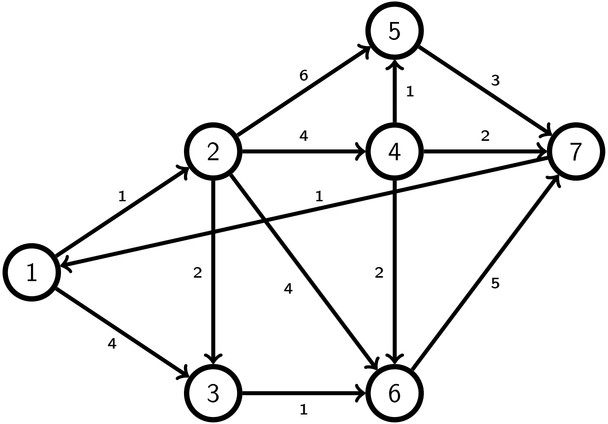
- If a label $d(i,i), i \in V$ takes a negative value
  - a cycle of negative length was found
  - algorithm cannot determine all shortest paths $\rightarrow$ stop
  - cycle can be determined using predecessor function $pred(.,.)$

Floyd-Warshall algorithm – pseudocode

Algorithm 3: Floyd-Warshall algorithm

```
// Initialization
SET d(i,i) := 0, pred(i,i) := undef for all i ∈ V
SET d(i,j) := c_ij, pred(i,j) := i for all (i,j) ∈ A
SET d(i,j) := ∞, pred(i,j) := undef for all i,j ∈ (V × V) \ A, i ≠ j
// Loop
for k ∈ V do
    for i ∈ V do
        for j ∈ V do
            if d(i,j) > d(i,k) + d(k,j) then
                SET d(i,j) := d(i,k) + d(k,j)
                SET pred(i,j) := pred(k,j)
        if d(i,i) < 0 then
            STOP, ∃ cycle with negative cost
// Output:  predecessors pred(·,·) and distances d(·,·)
```

Floyd-Warshall algorithm – example



Floyd-Warshall algorithm – example (Initialization)

Labels $d(i,j)$ and predecessors $pred(i,j)$

$$\begin{pmatrix} 0 & 1 & 4 & \infty & \infty & \infty & \infty \\ \infty & 0 & 2 & 4 & 6 & 4 & \infty \\ \infty & \infty & 0 & \infty & \infty & 1 & \infty \\ \infty & \infty & \infty & 0 & 1 & 2 & 2 \\ \infty & \infty & \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & 0 & 5 \\ 1 & \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix} \qquad \begin{pmatrix} \perp & 1 & 1 & \perp & \perp & \perp & \perp \\ \perp & \perp & 2 & 2 & 2 & 2 & \perp \\ \perp & \perp & \perp & \perp & \perp & 3 & \perp \\ \perp & \perp & \perp & \perp & 4 & 4 & 4 \\ \perp & \perp & \perp & \perp & \perp & \perp & 5 \\ \perp & \perp & \perp & \perp & \perp & \perp & 6 \\ 7 & \perp & \perp & \perp & \perp & \perp & \perp \end{pmatrix}$$

Floyd-Warshall algorithm – example (Iteration $k = 1$)

Labels $d(i, j)$ and predecessors $pred(i, j)$

$$\begin{pmatrix} 0 & 1 & 4 & \infty & \infty & \infty & \infty \\ \infty & 0 & 2 & 4 & 6 & 4 & \infty \\ \infty & \infty & 0 & \infty & \infty & 1 & \infty \\ \infty & \infty & \infty & 0 & 1 & 2 & 2 \\ \infty & \infty & \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & 0 & 5 \\ 1 & \underline{2} & \underline{5} & \infty & \infty & \infty & 0 \end{pmatrix} \qquad \begin{pmatrix} \perp & 1 & 1 & \perp & \perp & \perp & \perp \\ \perp & \perp & 2 & 2 & 2 & 2 & \perp \\ \perp & \perp & \perp & \perp & \perp & 3 & \perp \\ \perp & \perp & \perp & \perp & 4 & 4 & 4 \\ \perp & \perp & \perp & \perp & \perp & \perp & 5 \\ \perp & \perp & \perp & \perp & \perp & \perp & 6 \\ 7 & \underline{1} & \underline{1} & \perp & \perp & \perp & \perp \end{pmatrix}$$

Floyd-Warshall algorithm – example (Iteration $k = 2$)

Labels $d(i, j)$ and predecessors $pred(i, j)$

$$\begin{pmatrix} 0 & 1 & \underline{3} & \underline{5} & \underline{7} & \underline{5} & \infty \\ \infty & 0 & 2 & 4 & 6 & 4 & \infty \\ \infty & \infty & 0 & \infty & \infty & 1 & \infty \\ \infty & \infty & \infty & 0 & 1 & 2 & 2 \\ \infty & \infty & \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & 0 & 5 \\ 1 & 2 & \underline{4} & \underline{6} & \underline{8} & \underline{6} & 0 \end{pmatrix} \quad \begin{pmatrix} \perp & 1 & \underline{2} & \underline{2} & \underline{2} & \underline{2} & \perp \\ \perp & \perp & 2 & 2 & 2 & 2 & \perp \\ \perp & \perp & \perp & \perp & \perp & 3 & \perp \\ \perp & \perp & \perp & \perp & 4 & 4 & 4 \\ \perp & \perp & \perp & \perp & \perp & \perp & 5 \\ \perp & \perp & \perp & \perp & \perp & \perp & 6 \\ 7 & 1 & \underline{2} & \underline{2} & \underline{2} & \underline{2} & \perp \end{pmatrix}$$

Floyd-Warshall algorithm – example (Iteration $k = 3$)

Labels $d(i,j)$ and predecessors $pred(i,j)$

$$\begin{pmatrix} 0 & 1 & 3 & 5 & 7 & \underline{4} & \infty \\ \infty & 0 & 2 & 4 & 6 & \underline{3} & \infty \\ \infty & \infty & 0 & \infty & \infty & 1 & \infty \\ \infty & \infty & \infty & 0 & 1 & 2 & 2 \\ \infty & \infty & \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & 0 & 5 \\ 1 & 2 & 4 & 6 & 8 & \underline{5} & 0 \end{pmatrix} \qquad \begin{pmatrix} \perp & 1 & 2 & 2 & 2 & \underline{3} & \perp \\ \perp & \perp & 2 & 2 & 2 & \underline{3} & \perp \\ \perp & \perp & \perp & \perp & \perp & 3 & \perp \\ \perp & \perp & \perp & \perp & 4 & 4 & 4 \\ \perp & \perp & \perp & \perp & \perp & \perp & 5 \\ \perp & \perp & \perp & \perp & \perp & \perp & 6 \\ 7 & 1 & 2 & 2 & 2 & \underline{3} & \perp \end{pmatrix}$$

Floyd-Warshall algorithm – example (Iteration $k = 4$)

Labels $d(i,j)$ and predecessors $pred(i,j)$

$$\begin{pmatrix} 0 & 1 & 3 & 5 & \underline{6} & 4 & \underline{7} \\ \infty & 0 & 2 & 4 & \underline{5} & 3 & \underline{6} \\ \infty & \infty & 0 & \infty & \infty & 1 & \infty \\ \infty & \infty & \infty & 0 & 1 & 2 & 2 \\ \infty & \infty & \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & 0 & 5 \\ 1 & 2 & 4 & 6 & \underline{7} & 5 & 0 \end{pmatrix} \qquad \begin{pmatrix} \perp & 1 & 2 & 2 & \underline{4} & 3 & \underline{4} \\ \perp & \perp & 2 & 2 & \underline{4} & 3 & \underline{4} \\ \perp & \perp & \perp & \perp & \perp & 3 & \perp \\ \perp & \perp & \perp & \perp & 4 & 4 & 4 \\ \perp & \perp & \perp & \perp & \perp & \perp & 5 \\ \perp & \perp & \perp & \perp & \perp & \perp & 6 \\ 7 & 1 & 2 & 2 & \underline{4} & 3 & \perp \end{pmatrix}$$

Floyd-Warshall algorithm – example (Iteration $k = 5$)

Labels $d(i, j)$ and predecessors $pred(i, j)$ (no change)

$$\begin{pmatrix} 0 & 1 & 3 & 5 & 6 & 4 & 7 \\ \infty & 0 & 2 & 4 & 5 & 3 & 6 \\ \infty & \infty & 0 & \infty & \infty & 1 & \infty \\ \infty & \infty & \infty & 0 & 1 & 2 & 2 \\ \infty & \infty & \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & 0 & 5 \\ 1 & 2 & 4 & 6 & 7 & 5 & 0 \end{pmatrix} \qquad \begin{pmatrix} \perp & 1 & 2 & 2 & 4 & 3 & 4 \\ \perp & \perp & 2 & 2 & 4 & 3 & 4 \\ \perp & \perp & \perp & \perp & \perp & 3 & \perp \\ \perp & \perp & \perp & \perp & 4 & 4 & 4 \\ \perp & \perp & \perp & \perp & \perp & \perp & 5 \\ \perp & \perp & \perp & \perp & \perp & \perp & 6 \\ 7 & 1 & 2 & 2 & 4 & 3 & \perp \end{pmatrix}$$

Floyd-Warshall algorithm – example (Iteration $k = 6$)

Labels $d(i,j)$ and predecessors $pred(i,j)$

$$\begin{pmatrix} 0 & 1 & 3 & 5 & 6 & 4 & 7 \\ \infty & 0 & 2 & 4 & 5 & 3 & 6 \\ \infty & \infty & 0 & \infty & \infty & 1 & \underline{6} \\ \infty & \infty & \infty & 0 & 1 & 2 & 2 \\ \infty & \infty & \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & 0 & 5 \\ 1 & 2 & 4 & 6 & 7 & 5 & 0 \end{pmatrix} \qquad \begin{pmatrix} \perp & 1 & 2 & 2 & 4 & 3 & 4 \\ \perp & \perp & 2 & 2 & 4 & 3 & 4 \\ \perp & \perp & \perp & \perp & \perp & 3 & \underline{6} \\ \perp & \perp & \perp & \perp & 4 & 4 & 4 \\ \perp & \perp & \perp & \perp & \perp & \perp & 5 \\ \perp & \perp & \perp & \perp & \perp & \perp & 6 \\ 7 & 1 & 2 & 2 & 4 & 3 & \perp \end{pmatrix}$$

Floyd-Warshall algorithm – example (Iteration $k = 7$)

Labels $d(i,j)$ and predecessors $pred(i,j)$

$$\begin{pmatrix} 0 & 1 & 3 & 5 & 6 & 4 & 7 \\ \underline{7} & 0 & 2 & 4 & 5 & 3 & 6 \\ \underline{7} & \underline{8} & 0 & \underline{12} & \underline{13} & 1 & 6 \\ \underline{3} & \underline{4} & \underline{6} & 0 & 1 & 2 & 2 \\ \underline{4} & \underline{5} & \underline{7} & \underline{9} & 0 & \underline{8} & 3 \\ \underline{6} & \underline{7} & \underline{9} & \underline{11} & \underline{12} & 0 & 5 \\ 1 & 2 & 4 & 6 & 7 & 5 & 0 \end{pmatrix} \quad \begin{pmatrix} \perp & 1 & 2 & 2 & 4 & 3 & 4 \\ \underline{7} & \perp & 2 & 2 & 4 & 3 & 4 \\ \underline{7} & \underline{1} & \perp & \underline{2} & \underline{4} & 3 & 6 \\ \underline{7} & \underline{1} & \underline{2} & \perp & 4 & 4 & 4 \\ \underline{7} & \underline{1} & \underline{2} & \underline{2} & \perp & \underline{3} & 5 \\ \underline{7} & \underline{1} & \underline{2} & \underline{2} & \underline{4} & \perp & 6 \\ 7 & 1 & 2 & 2 & 4 & 3 & \perp \end{pmatrix}$$

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, New Jersey, 1993. ISBN 0-13-617549-X.