Logistics systems planning I
Optimization of logistics systems
Transportation planning – Shortest path problems

Univ.-Prof. Dr. Michael Schneider

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

schneider@dpo.rwth-aachen.de

Course agenda

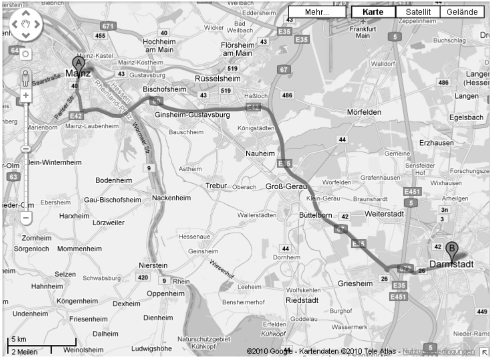
Goals of the section:

- classify types of shortest path problems
- know LP formulations
- understand optimality conditions and a generic shortest path algorithm
- know prerequisites of different algorithms
- manually carry out the algorithms

Agenda

1. Shortest path problems – Part I
   - Problem definition and applications
   - Network flow models
   - A generic shortest path algorithm
   - Pulling and reaching algorithm

Shortest path problem – real



,ÄúWhich is the shortest/fastest/cheapest path from A to B,Äù

Shortest path problems – Applications

- Route planners: software or web services to plan trips
- Logistics systems: efficient design of distribution systems (shortest or fastest route in a transportation system)
- Base data: geographical information systems (GIS) provide distance matrices as input data for route planning. Knowledge of spatial and temporal distances is a prerequisite for the majority of advanced planning problems in transportation
- Problem structure: Shortest path problems as subproblems or auxiliary problems in many other optimization problems (e.g., vehicle routing, lot sizing, ...)

Weight or length of a walk

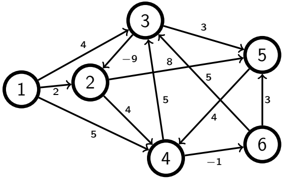
Length of a walk

In a weighted digraph (V, A, c_{ij}), the length of a walk $P = (v_1, a_1, v_2, a_2, \ldots, v_p)$ is defined as

$$c(P) := \sum_{i=1}^{p-1} c_{a_i} = \sum_{i=1}^{p-1} c_{v_i, v_{i+1}}.$$

- Negative weights natural in many applications: revenue/costs, column generation, Lagrangian relaxation
- If some weights $c_{ij} < 0$ (negative), walks with $c(P) < 0$ can occur. It is then possible that no walk P with minimum length between s and t exists.

Example: Cycles of negative length



- Cycle $C_1 = (2, 4, 3, 2)$ with length 0
- Cycle $C_2 = (2, 4, 6, 3, 2)$ with length -1, i.e., cycle of negative length
- No shortest walk from $s = 1$ to $t = 6$ exists!
  - walk $K_1 = (1, 4, 6)$ has length 4.
  - walk $K_2 = (1, \underline{4, 6, 3, 2, 4}, 6)$ has length $4 - 1 = 3$
  - walk $K_3 = (1, \underline{4, 6, 3, 2, 4, 6, 3, 2, 4}, 6)$ has length $4 - 2 = 2$
  - walk $K_4 = (1, \underline{4, 6, 3, 2, 4, 6, 3, 2, 4, 6, 3, 2, 4}, 6)$ has length $4 - 3 = 1$

  etc.

Cycles of negative length

- Finite digraphs contain a finite number of paths $\rightarrow$ the problem of finding a shortest path is always well-defined.
- Example on last slide: $(1, 3, 2, 4, 6)$ is a shortest 1-6-path with length -2.

Shortest path – problem variants

- Different variants: Find (a) shortest path(s)
  - between node s and node t (s and t given)
  - between node s and all other nodes in the graph $t \neq s$ (s given)
  - between all pairs of nodes (s, t)
- Algorithms to solve shortest path problems have different prerequisites:
  - arbitrary vs. non-negative arc weights
  - digraphs with vs. without cycles
  - existence of cycles of negative length

Agenda

SPP models – problem definition

- Find a shortest (cost-minimal) path between two nodes in a graph
- Given:
    - weighted digraph $D = (V, A, c)$ with length c_{ij} for all arcs $(i, j) \in A$
    - start node (source) $s \in V$
    - destination node (=sink) $t \in V$
- We seek:
    - the information which arcs (i, j) are contained in the shortest path $\rightarrow$ decision variables $x_{ij} \in \{0, 1\}$
    - equivalent to $0 \leq x_{ij} \leq 1$ or $x_{ij} \geq 0$
- Prerequisite for following models: no cycles of negative length in $D = (V, A, c)$ $\rightarrow$ follows directly from non-negative arc weights $c_{ij} \geq 0$ or if the digraph is acyclic

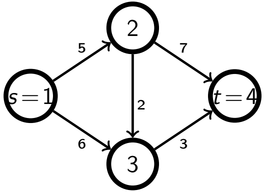
Shortest path problem – model I

- Idea: transshipment problem with
  - start $s \in V$ with supply of 1 unit
  - destination node $t \in V$ with demand of 1 unit
  - all other nodes are transshipment nodes; no capacity constraints

$$
\begin{aligned}
z_{SP} \quad &= \quad \min \sum_{(i,j) \in A} c_{ij} x_{ij} && (1) \\
\text{s.t.} \quad & \sum_{(i,j) \in \delta^+(i)} x_{ij} - \sum_{(h,i) \in \delta^-(i)} x_{hi} = \begin{cases} +1 & i = s \\ -1 & i = t \\ 0 & s, t \neq i \in V \end{cases} && (2) \\
& x_{ij} \geq 0 \text{ (or } \in \{0,1\}) \quad \forall (i,j) \in A && (3)
\end{aligned}
$$

(1) Minimize length of path; (2) flow conservation; (3) NNC

Example model I



All s-t-paths [length]:

Model (explicit):

Shortest path problem – model II

- Model to determine simultaneously all shortest paths from a given node $s \in V$ to all other nodes $v \in V, v \neq s$
- Prerequisite: no cycles of negative length and $D = (V, A)$ is connected (each node is reachable from s)

Model:

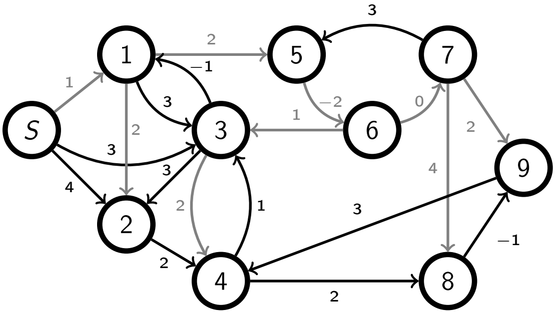
$$\min \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{4}$$

$$\text{s.t.} \quad \sum_{(i,j) \in \delta^+(i)} x_{ij} - \sum_{(h,i) \in \delta^-(i)} x_{hi} = \begin{cases} |V| - 1 & i = s \\ -1 & i \in V, i \neq s \end{cases} \tag{5}$$

$$x_{ij} \geq 0 \text{ (bzw. } \in \mathbb{Z}_+) \quad \forall (i,j) \in A \tag{6}$$

Shortest path problem – Solution model II

Solution for source s and resulting shortest paths tree:



Question: Which values do the x_{ij} of the blue arcs take?
$x_{s1} = 9, x_{12} = 1, x_{15} = 7, x_{56} = 6, x_{63} = 2 \ldots x_{79} = 1$

Shortest paths tree

- In a shortest path tree, the unique path from s to node j corresponds to the sought-after shortest path from s to j
- Node $pred(j)$ is the predecessor node of node j in the corresponding shortest path
- To determine all shortest paths from source s to all nodes of the digraph, knowledge of the predecessor node $pred(j)$ for $j \in V \setminus \{s\}$ is sufficient. The shortest path to node j results from ‚Äúreading backwards from destination to start‚Äù by means of the predecessor function $pred(\cdot)$
- Start sequences of shortest paths are shortest paths themselves

Agenda

Shortest path algorithms

- Different prerequisites
  - arbitrary vs. non-negative arc weights
  - digraphs with vs. without cycles
  - existence of cycles of negative length $\rightarrow$ in the following, we assume that such cycles do not occur
- Algorithms
  - Pulling and reaching algorithm for digraphs without cycles
  - Dijkstra algorithm for digraphs with non-negative arc weights, i.e., $c_{ij} \geq 0$ for all $(i, j) \in A$
  - FIFO algorithm for arbitrary digraphs
  - Floyd-Warshall algorithm to determine shortest paths between all pairs of nodes for arbitrary digraphs

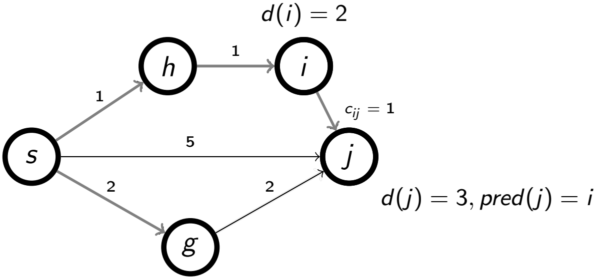
Labeling algorithms I

- Labeling algorithms are often used to solve shortest path problems
- We first consider s-to-all shortest path problems
- For each node $i \in V$, a label $d(i)$ is
  - the length of an (arbitrary) path from s to i, or
  - an upper bound for the length of such a path (also $d(i) = \infty$ is allowed)

Labeling algorithms II

If the inequality $d(j) > d(i) + c_{ij}$ holds for an arc $(i, j) \in A$, we use the following update to get a new label for node j

$$\begin{aligned} d(j) &:= d(i) + c_{ij} \\ pred(j) &:= i \end{aligned}$$



Labeling algorithms III

- Labeling algorithms are iterative procedures
- Start with upper bounds $d(j)$ for the lengths of shortest paths from the source s to the nodes j
- Successively decrease the values $d(j)$ ($\rightarrow$ update) until they correspond to the actual length of a shortest path
- It holds $d(s) := 0$, and, in general, $d(i) := \infty$ for $i \in V, i \neq s$ at the beginning

Path optimality conditions I

Path optimality conditions

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

- for each node $v \in V$, $d(v)$ is the length of a shortest path from s to v, **iff**
- for all arcs $(i, j) \in A$, the labels satisfy the path optimality conditions

$$d(j) \leq d(i) + c_{ij}.$$

Path optimality conditions

Conditions (Path-Opt) are optimality conditions for labels $d(j)$, i.e., their validity implies that the labels specify the lengths of the shortest paths.

Proof: Assume that conditions (Path-Opt) are fulfilled for labels $d(j)$. We have to show that each label $d(j)$ is also a lower bound for the length of an arbitrary path from s to j. For such a path $P = (s = i_0, i_1, \ldots, i_L = j)$, the following holds:

$$
\begin{aligned}
c(P) &= \sum_{k=1}^{L} c_{i_{k-1},i_k} \\
&\geq \sum_{k=1}^{L} (d(i_k) - d(i_{k-1})) \\
&= d(i_L) - d(i_0) = d(j) - d(s) = d(j).
\end{aligned}
$$

Because path P can also be a shortest path, $d(j)$ is an upper and lower bound for the length of a shortest path at the same time.

Generic shortest path algorithm

Algorithm 1: Generic algorithm

```
// Initialization
SET d(s) := 0, pred(s) := undef
SET d(j) := ∞ for all j ∈ V \ {s}
// Loop
while optimality conditions violated do
    SELECT arc (i,j) ∈ A with d(j) > d(i) + c_ij
    SET d(j) := d(i) + c_ij
    SET pred(j) := i
// Output:  predecessor pred(·) and distances d(·)
```

Different shortest path algorithms only differ in the selection rule for arcs. Typically:

- outer loop over nodes $i \in V$
- inner loop over all arcs $(i, j) \in \delta^+(i)$ [or $(j, i) \in \delta^-(i)$]

Label setting vs. label correcting

- Label setting algorithms (LSA)
  - in each iteration (outer loop) the label of one node is fixed to its final value. This is achieved by means of the node selection rule
  - potentially several labels are modified within one iteration of the outer loop
  - prerequisites: acyclic graph or non-negative arc weights
- Label correcting algorithms (LCA)
  - can modify all labels multiple times until the optimality conditions simultaneously hold for all arcs $(i, j) \in A$
  - often use simpler selection rules compared to LSA $\rightarrow$ potentially better average-case performance, but usually inferior concerning worst-case performance
  - LCA have no prerequisites concerning the weighted digraph (besides that no cycles of negative length are allowed)

Agenda

1. Shortest path problems – Part I
   - Problem definition and applications
   - Network flow models
   - A generic shortest path algorithm
   - Pulling and reaching algorithm

Acyclic graphs and topological sorting I

- In acyclic digraphs, nodes can be presorted such that label setting algorithms to solve the shortest path problem with source s require $\mathcal{O}(|A|)$ steps

Topological sorting

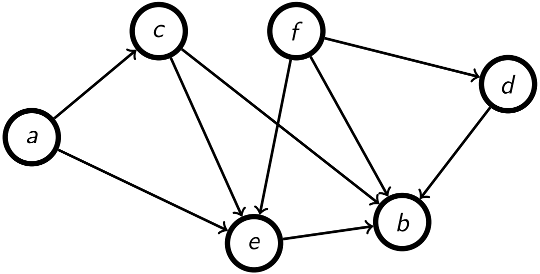
A sorting $(v_1, v_2, \ldots, v_m)$ of all nodes of digraph D is called topological sorting if only arcs from a node with smaller index to a node with larger index exist. Formally:

$$(v_i, v_j) \in A \quad \Rightarrow \quad i < j.$$

Acyclic graphs and topological sorting II

- Iterative procedure to generate topological sorting
  1. remove randomly one of the nodes without predecessor together with all incident arcs
  2. repeat until no node can be removed anymore
- In an acyclic graph, the procedure ends with an empty graph (with 0 nodes). Otherwise a subgraph exists, in which each node has at least one predecessor.
- A digraph can be topologically sorted iff it is acyclic

Acyclic graphs and topological sorting: Example



Topological sorting:

Acyclic graphs and topological sorting III

- Assume that nodes $V = \{1, 2, \ldots, m\}$ are already topologically sorted, and the labels $d(i)$ with $i = 1, ..., k$ of the first k nodes satisfy the optimality conditions. Then:
  - the labels $d(i)$ of $i = 1, ..., k$ are not modified anymore because $d(j) > d(i) + c_{ij}$ can only hold for nodes $j > k$
  - only labels for nodes $k + 1, \ldots, m$ must be set
  - in particular: for node $j = k + 1$, all potential predecessors possess their final evaluation
- These observations lead directly to the so-called pulling algorithm

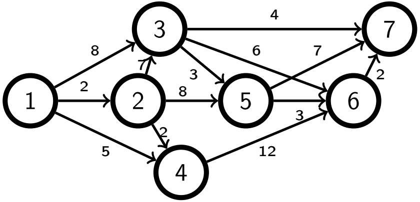
Pulling algorithm

Nodes $V = \{s = 1, 2, \ldots, m\}$ are already topologically sorted as $(s = 1, 2, \ldots, m)$.

Algorithm 2: Pulling algorithm

`// Initialization`
SET $d(s) := 0$, $pred(s) := undef$
`// Loop`
for $j = 2, \ldots, m$ **do**
  SELECT arc $(i, j) \in \delta^-(j)$ with $d(i) + c_{ij}$ minimum
  SET $d(j) := d(i) + c_{ij}$
  SET $pred(j) := i$
`// Output: predecessor` $pred(\cdot)$ `and distances` $d(\cdot)$

Pulling algorithm: Example



Node j	Predecessor $\delta^-(j)$	$d(i) + c_{ij}$	$d(j)$	$pred(j)$
(Initialization)				
1	–		$d(1) = 0$	$pred(1) = undef$
(Loop)				
2	$(1, 2)$	2	$d(2) = 2$	$pred(2) = 1$
3	$(1, 3), (2, 3)$	8, 9	$d(3) = 8$	$pred(3) = 1$
4	$(1, 4), (2, 4)$	5, 4	$d(4) = 4$	$pred(4) = 2$
5	$(2, 5), (3, 5)$	10, 11	$d(5) = 10$	$pred(5) = 2$
6	$(3, 6), (4, 6), (5, 6)$	14, 16, 13	$d(6) = 13$	$pred(6) = 5$
7	$(3, 7), (5, 7), (6, 7)$	12, 17, 15	$d(7) = 12$	$pred(7) = 3$

Reaching algorithm

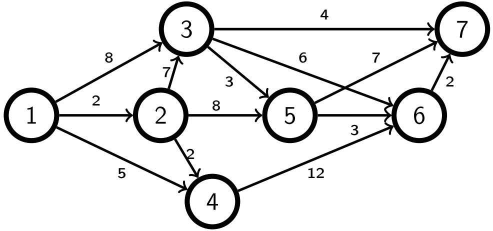
Nodes $V = \{s = 1, 2, \dots, m\}$ are already topologically sorted as $(s = 1, 2, \dots, m)$.

Algorithm 3: Reaching algorithm

// Initialization
SET $d(s) := 0$, $pred(s) := undef$
SET $d(j) := \infty$ for all $j \in V \setminus \{1\}$
// Loop
for $i = 1, \dots, m-1$ **do**
  for $(i, j) \in \delta^+(i)$ **do**
    if $d(j) > d(i) + c_{ij}$ **then**
      SET $d(j) := d(i) + c_{ij}$
      SET $pred(j) := i$

// Output: predecessor $pred(\cdot)$ and distances $d(\cdot)$

Reaching algorithm: Example



Node i	Successor $\delta^+(i)$	$d(i) + c_{ij}$	modified $d(j)$	$pred(j)$
(Initialization)			$d(1) = 0$	$pred(1) = undef$
(Loop)				
1	$(1,2), (1,3), (1,4)$	2, 8, 5	$d(2) = 2, d(3) = 8, d(4) = 5$	$pred(2) = 1,\ pred(3) = 1,\ pred(4) = 1$
2	$(2,3), (2,4), (2,5)$	9, 4, 10	$d(4) = 4, d(5) = 10$	$pred(4) = 2,\ pred(5) = 2$
3	$(3,5), (3,6), (3,7)$	11, 14, 12	$d(6) = 14, d(7) = 12$	$pred(6) = 3,\ pred(7) = 3$
4	$(4,6)$	16		
5	$(5,6), (5,7)$	13, 17	$d(6) = 13$	$pred(6) = 5$
6	$(6,7)$	15		
7	$-$			