MVE165/MMG630, Applied Optimization
Lecture 8
Shortest paths and network flow models; linear programming formulation of flows in networks

Ann-Brith Strömberg

2012–04–17

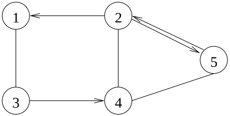
Network models—examples (Ch. 8)

Many different problems can be formulated as graph or network flow models:

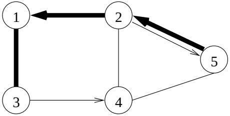
- Find the shortest/fastest connection from Johanneberg to Lindholmen
- Connect a number of base stations minimizing the total cost of construction
- Find the maximum capacity in a given water pipeline network
- Find a time schedule (start and completion times) for activities in a project
- Find how much goods should be transported from each supplier to each point of demand, using which links in a transport system
- ...

Definitions and terminology

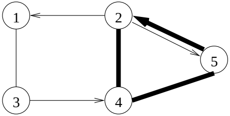
- A *graph* consists of a set N of *nodes* linked by a set E of (undirected) *edges* and/or a set A of (directed) *arcs*



- For many applications: distances (or costs) d_{ij} on the arcs/edges
- A *path* is a sequence of arcs between two nodes

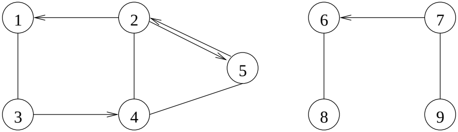


- A *cycle/loop* is a path that connects a node to itself



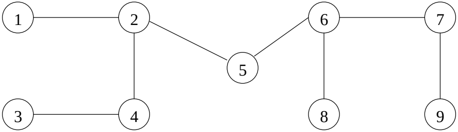
Definitions and terminology

- A *connected graph* has at least one path between each pair of nodes (example: an unconnected graph)



- A *tree/forest* is a graph without cycles connecting a *subset* of the nodes.

- A *spanning tree* is a tree that connects *all* the nodes of a graph



The minimum spanning tree (MST) problem

- Given an undirected graph $G = (N, E)$ with nodes N, edges E and distances d_{ij} for each edge $(i, j) \in E$
- Find a subset of the edges that connects all nodes at minimum total distance
- The number of edges in a spanning tree is $|N| - 1$
- A (spanning) tree contains *no cycles*
- MST is a very simple problem (a matroid) that can be solved by *greedy algorithms*

Greedy algorithms for MST

- Prim's algorithm
  1. Start at an arbitrary node
  2. Among the nodes that are not yet connected, choose the one that can be connected at minimum cost
  3. Stop when all nodes are connected
- Kruskal's algorithm
  1. Sort the edges by increasing distances
  2. Choose edges starting from the beginning of the list; skip edges resulting in cycles
  3. Stop when all nodes are connected
- Solve an example!

The shortest path problem (Ch. 8.4)

- Given: a network of nodes N, (directed) arcs A, and arc distances d_{ij}, $(i, j) \in A$
- Find the shortest path from a source node ($s \in N$) to a destination node ($t \in N$)

Examples that can be formulated as shortest path problems:

- Find the shortest connection from Johanneberg to Lindholmen (using bus, tram, bike, car, or combinations, ...)
- Find most reliable route (failure probabilities for the arcs)
- Find the shortest routes for data on the internet
- ...

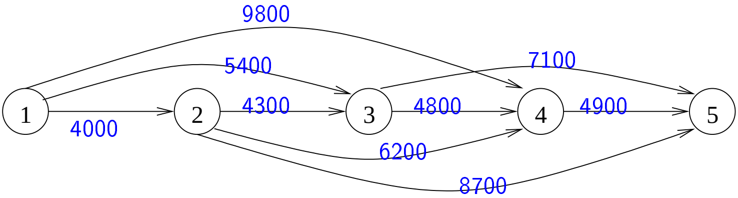
Example: Equipment replacement

- RentCar wants to find a replacement strategy for its cars for a 4-year planning period
- Each year, a car can be kept or replaced
- The replacement cost for each year and period is given in the table below
- Each car should be used at least 1 year and at most 3 years

Equipment obtained at start of year	Replacement cost for # years in operation: 1	2	3
1	4000	5400	9800
2	4300	6200	8700
3	4800	7100	—
4	4900	—	—

Example: Equipment replacement

Equipment obtained at start of year	Replacement cost for # years in operation		
	1	2	3
1	4000	5400	9800
2	4300	6200	8700
3	4800	7100	—
4	4900	—	—



Cheapest path from 1 to 5: 1 → 3 → 5. Cost: 12500

A linear programming formulation: shortest path from node $s \in V$ to node $t \in V$

- For each arc $(i,j) \in A$, let x_{ij} be the flow on the arc
- *Flow balance in each node $k \in N$*
- $x_{ij} = 1$ if arc (i,j) is in the shortest path and $x_{ij} = 0$ otherwise
- Linear programming formulation (assume $d_{ij} \geq 0$):

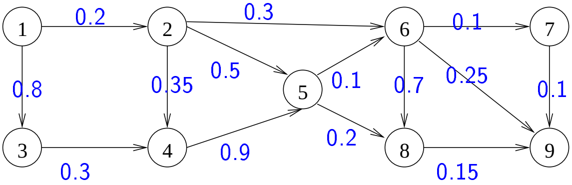
$$
\begin{aligned}
\min \quad & \sum_{(i,j)\in A} d_{ij} x_{ij}, \\
\text{s.t.} \quad & \sum_{i:(i,k)\in A} x_{ik} - \sum_{j:(k,j)\in A} x_{kj} = \begin{cases} -1, & k = s, \\ 1, & k = t, \\ 0, & k \in N \setminus \{s,t\}, \end{cases} \\
& x_{ij} \geq 0, \quad (i,j) \in A.
\end{aligned}
$$

- Linear programming dual:

$$
\begin{aligned}
\max \quad & y_t - y_s, \\
\text{s.t.} \quad & y_j - y_i \leq d_{ij}, \quad (i,j) \in A \\
& y_k \quad \text{free}, \quad k \in N
\end{aligned}
$$

Example: Most reliable route

- Mr Q drives to work daily
- All road links he can choose for a path to work are patrolled by the police
- It is possible to assign a probability p_{ij} of *not* being stopped by the police on link (i, j)
- Mr Q wants to find the "shortest" (safest?) path in the sense that the probability of being stopped is as low as possible
- maximize $Prob$(not being stopped)



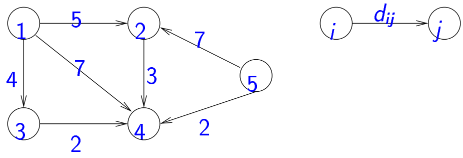
- Ex. $1 \longrightarrow 4$: $\max\{p_{12}p_{24}; p_{13}p_{34}\} = \max\{0.2 \cdot 0.35; 0.8 \cdot 0.3\}$
- Note: This version *cannot* be formulated as a linear program

Discrete dynamic programming methods (Ch. 18)

- Efficient methods for shortest path problems (and some other models)
- Expecially to find shortest paths from *many* to *many* nodes
- Linear programming can be used but is less efficient
- Functional notation
  - y_j = length of shortest (most reliable) path from source node (s) to node j
  - $y_k = \infty$ if no path exists
  - $x_{ij}^k = \begin{cases} 1 & \text{if arc/edge } (i,j) \text{ is part of the optimal} \\ & \text{path from source node } s \text{ to node } k \\ 0 & \text{otherwise} \end{cases}$

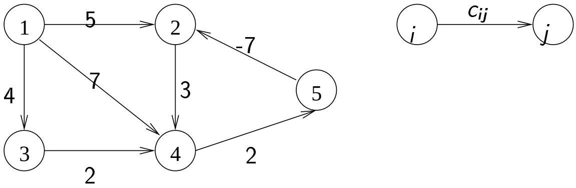
Example: shortest paths (Ch. 8.4)

- Shortest paths from node 1 to all other nodes



- $y_1 = 0,\ y_2 = 5,\ y_3 = 4,\ y_4 = 6,\ y_5 = \infty$
- $x^1_{12} = x^1_{13} = x^1_{14} = x^1_{24} = x^1_{34} = x^1_{52} = x^1_{54} = 0$
- $x^2_{12} = 1,\ x^2_{13} = x^2_{14} = x^2_{24} = x^2_{34} = x^2_{52} = x^2_{54} = 0$
- $x^3_{13} = 1,\ x^3_{12} = x^3_{14} = x^3_{24} = x^3_{34} = x^3_{52} = x^3_{54} = 0$
- $x^4_{13} = x^4_{34} = 1,\ x^4_{12} = x^4_{14} = x^4_{24} = x^4_{52} = x^4_{54} = 0$
- No path exists from 1 to 5
- The arcs in the shortest paths from one node to all other (reachable) nodes forms a tree ($(1,2)$, $(1,3)$, and $(3,4)$)
- If all nodes are reachable: shortest path tree is a spanning tree

Negative cycles



- A *negative cycle* is a cycle of *negative* total length

$\Rightarrow$ Shortest path "length" $\rightarrow -\infty$

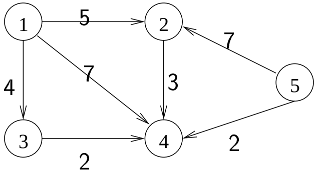
$\Rightarrow$ Dynamic programming algorithms do usually not apply

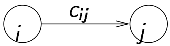
Functional equations (Bellman's equations)

- Principle of optimality: In a graph with no negative cycles, optimal paths have optimal subpaths

$\Rightarrow$ Functional equations for shortest path from node s to all other nodes in a graph with *no negative cycles*

  - $y_s = 0$
  - $y_j = \min\{y_i + c_{ij} : \text{ arc/edge } (i, j) \text{ exists } \}$ for all $j \neq s$





Variants of functional equations

- Most reliable path (failure probability $p_{ij} \in [0,1]$ for arc (i,j)):
  - $y_s = 1$
  - $y_j = \max\left\{ y_i \cdot p_{ij} : \text{ arc/edge } (i,j) \text{ exists } \right\}$ for all $j \neq s$
- Highest capacity path (capacity $K_{ij} \geq 0$ on arc (i,j)):
  - $y_s = \infty$
  - $y_j = \max\left\{ \min\{y_i; K_{ij}\} : \text{ arc/edge } (i,j) \text{ exists } \right\}, \ j \neq s$
- Paths from all nodes to all other nodes in a graph with no negative cycles (arc distances d_{ij}):
  - $y_{jj} = 0$ for all j
  - $y_{j\ell} = \min\left\{ d_{j\ell}; \{ y_{ji} + y_{i\ell} : i \neq j, \ell \} \right\}$ for all $j \neq \ell$

Algorithms for the shortest path problem: Dijkstra (Ch. 8.4.2)

- Find the shortest path between node s and node i when all arcs distances are non-negative
- N = set of all nodes; source node $s \in N$
- d_{ij} = distance on link from i to j for all $i, j \in N$
- $d_{ij} = \infty$ if no direct link from i to j

Step 0: $S := \{s\}$, $\bar{S} := N \setminus \{s\}$, and $y_i := d_{si}$, $i \in N$

Step 1:

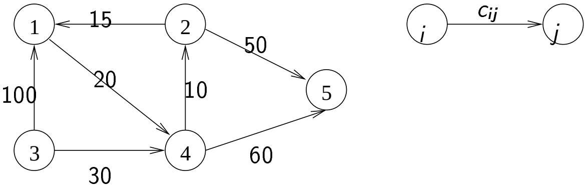
(a) If $\bar{S} = \emptyset$, stop. Else find node j such that $y_j = \min_{i \in \bar{S}} y_i$
$S := S \cup \{j\}$ and $\bar{S} := \bar{S} \setminus \{j\}$

(b) For all $k \in \bar{S}$ and $i \in S$:
If $y_k > y_i + d_{ik}$ set $y_k := y_i + d_{ik}$ and $pred(k) := i$

- The vector *pred* keeps track of the predecessors
- Dijkstra's algorithm actually finds shortest paths from the source to *all* others nodes

Example: Dijkstra's algorithm

Find the shortest path from node 1 to all other nodes (Homework)



Algorithms for the shortest path problem: Floyd–Warshall (Ch. 8.4.2)

- Computes shortest paths between each pair of nodes
- Negative distances are allowed but no negative cycles—but these can be detected
- Idea: Three nodes i, k, j and distances c_{ik}, c_{kj}, and c_{ij}
- $i \rightarrow k \rightarrow j$ is a short-cut if $c_{ik} + c_{kj} < c_{ij}$
- In each iteration $1 \ldots k$, check whether c_{ij} can be improved by using the short-cut via k
- Administration of the algorithm: Maintain two matrices per iteration: $D[k]$ for the distances and $pred[k]$ to keep track of the predecessor of each node

Floyd–Warshall's algorithm

Step 0: Initialize $D[0]$ and $pred[0]$

Step k:
- $D[k] := D[k-1]$, $pred[k] := pred[k-1]$
  For each element d_{ij} in $D[k]$:
  If $d_{ik} + d_{kj} < d_{ij}$, set $d_{ij} := d_{ik} + d_{kj}$ and $pred_{ij}[k] := k$
  Set $k := k + 1$
  If $k > n$ stop, else repeat Step k

Find the shortest path from node 3 to all other nodes