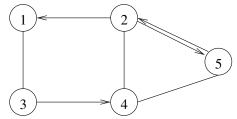
MVE165/MMG630, Applied Optimization
Lecture 5
Shortest paths and network flow models

Ann-Brith Strömberg

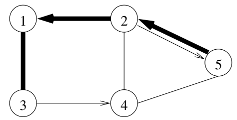
2009–03–24

Definitions and terminology

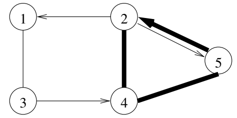
- A *graph* consists of a set N of *nodes* linked by a set A of (undirected) *edges* and/or (directed) *arcs*



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



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



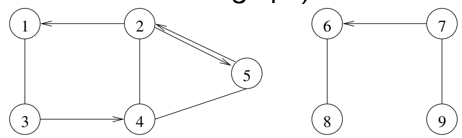
Network models—examples

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

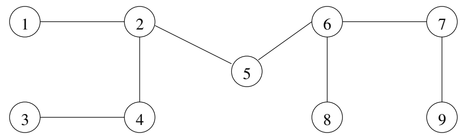
- Find the shortest/fastest connection from Johanneberg to Lindholmen
- Connect a number of base stations minimizing the total cost
- Find the maximum capacity in a given water pipeline network
- Find a time schedule (start and completion times) for activities in a project
- Find how much goods should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

Definitions and terminology

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



- A *tree* is a connected graph without cycles connecting a *subset* of the nodes.
- A *spanning tree* is a tree that connects *all* the nodes of a graph



The shortest path problem

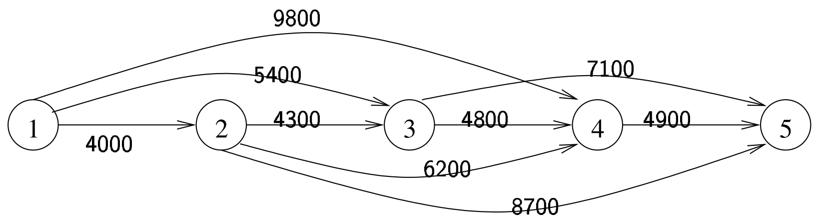
- Given: a network of nodes, arcs, and arc distances
- Find the shortest path from a source node to a destination node

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
- Solve the three-jug puzzle (three buckets 8, 5, and 3 liters)
- ...

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: $1 \to 3 \to 5$. Cost: 12500

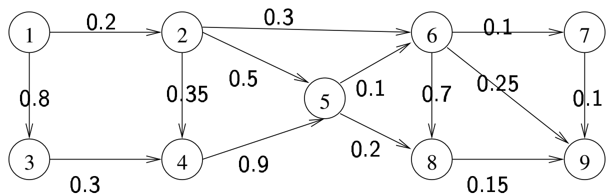
Example: Equipment replacement

- 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: 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)
- He wants to find the "shortest" (safest?) path in the sense that the probability of being stopped is as low as possible
- $\text{maximize}\ P(\text{not being stopped})$

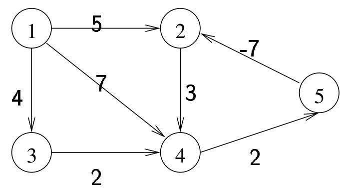


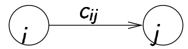
- Ex. $1 \to 4$: $\max\{p_{12} \cdot p_{24}; p_{13} \cdot p_{34}\} = \max\{0.2 \cdot 0.35; 0.8 \cdot 0.3\}$

Discrete dynamic programming methods

- 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
  - $v[k]$ = length of shortest (most reliable) path from source node to node k
  - $v[k] = \infty$ if no path exists
  - $x_{ij}[k] = \begin{cases} 1 & \text{if arc/edge } (i,j) \text{ is part of the optimal path from source node to node } k \\ 0 & \text{otherwise} \end{cases}$

Negative cycles





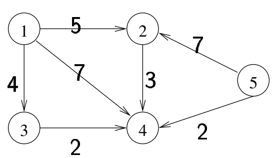
- A *negative cycle* is a cycle of *negative* total length

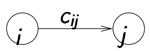
$\Rightarrow$ Shortest path "length" $\to -\infty$

$\Rightarrow$ Dynamic programming algorithms do usually not apply

Example: shortest paths

- Shortest paths from node 1 to all other nodes





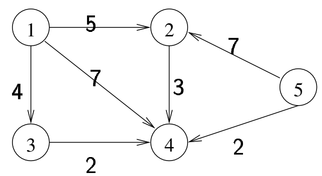
- $v[1] = 0,\ v[2] = 5,\ v[3] = 4,\ v[4] = 6,\ v[5] = \infty$
- $x_{12}[1] = x_{13}[1] = x_{14}[1] = x_{24}[1] = x_{34}[1] = x_{52}[1] = x_{54}[1] = 0$
- $x_{12}[2] = 1,\ x_{13}[2] = x_{14}[2] = x_{24}[2] = x_{34}[2] = x_{52}[2] = x_{54}[2] = 0$
- $x_{13}[3] = 1,\ x_{12}[3] = x_{14}[3] = x_{24}[3] = x_{34}[3] = x_{52}[3] = x_{54}[3] = 0$
- $x_{13}[4] = x_{34}[4] = 1,\ x_{12}[4] = x_{14}[4] = x_{24}[4] = x_{52}[4] = x_{54}[4] = 0$
- 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

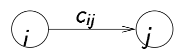
Functional equations

- 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*

  - $v[s] = 0$
  - $v[k] = \min\{v[i] + c_{ik} :\ \text{arc/edge } (i,k) \text{ exists }\}$ for all $k \neq s$



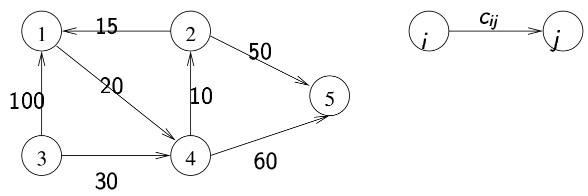


Variants of functional equations

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

Example: Dijkstra's algorithm

Find the shortest path from node 1 to all other nodes



Algorithms for the shortest path problem: Dijkstra

Dijkstra's algorithm finds the shortest path between a source node s and node i if all distances on the arcs are non-negative.

- N = set of all nodes,
- Source node $s \in N$
- c_{ij} = distance on link from i to j for all $i, j \in N$
- $c_{ij} = \infty$ if no direct link from i to j

Step 0: $S := \{s\}$, $\bar{S} := N \setminus \{s\}$, and $v[i] := c_{si}$, $i \in N$

Step 1: (a) If $\bar{S} = \emptyset$, stop. Else find node k such that $v[k] = \min_{i \in \bar{S}} v[i]$
$S := S \cup \{k\}$ and $\bar{S} := \bar{S} \setminus \{k\}$
(b) For all $j \in \bar{S}$ and $i \in S$:
If $v[j] > v[i] + c_{ij}$ set $v[j] := v[i] + c_{ij}$ and $pred(j) := i$

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

Algorithms for the shortest path problem: Floyd–Warshall

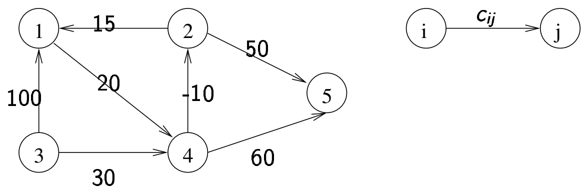
- Floyd's algorithm computes shortest paths between each pair of nodes
- Negative distances are allowed but no negative cycles—but these can be detected
- Idea: Three nodes i, k, j and distances c_{ik}, c_{kj}, and c_{ij}.
- $i \to k \to j$ is a short-cut if $c_{ik} + c_{kj} < c_{ij}$
- In each iteration $1 \ldots k$, check whether c_{ij} can be improved by using the short-cut via k.
- Administration of the algorithm: Maintain two matrices per iteration, C_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



A network formulation of the shortest path problem

Find the shortest path from node s to node t:
- Let for each arc x_{ij} be the flow on arc (i, j)
- $x_{ij} = 1$ if arc (i, j) is in the shortest path and $x_{ij} = 0$ otherwise
- Linear programming formulation:

$$\begin{array}{lrcll} \min & \sum_{(i,j)\in A} d_{ij}x_{ij}, & & & \\ \text{s.t.} & \sum_j x_{sj} & = & 1, & \\ & \sum_j x_{jt} & = & 1, & \\ & \sum_i x_{ik} - \sum_j x_{kj} & = & 0, & k \neq s, t, \quad (*) \\ & x_{ij} & \geq & 0. & \end{array}$$

- The constraints $(*)$ are *flow balance constraints*

Definition of general network flow problems

- A network consist of a set N of *nodes* linked by a set A of *arcs*
- A distance d_{ij} is associated with each arc
- Each node i in the network has a net demand b_i
- Each arc has an (unknown) amount of flow x_{ij} that is restricted by a maximum capacity $u_{ij} \in [0, \infty]$
- The flow through each node must be *balanced*
- A *graph* is a special case of a network
- A network flow problem can be formulated as a linear program

Maximum flow models

- Consider a district heating network with pipelines that transports energy from a number of sources to a number of destinations
- The network has several branches and intersections
- Pipe segment (i, j) has a maximum capacity of K_{ij} units of flow per time unit
- A pipe can be one- or bidirectional
- What is the maximum total amount of flow per time unit through this network?
- Another application of maximum flow models is evacuation of buildings

Linear programming formulation of maximum flow problem

- Graph: $G = (V, A, K)$ (nodes, directed arcs, arc capacities)

$$
\begin{array}{llrcll}
\text{[Primal]} & \max & & & v, & \\
& \text{s.t.} & -\sum_{j:(s,j)\in A} x_{sj} + v & = & 0, & \\
& & \sum_{j:(j,t)\in A} x_{jt} - v & = & 0, & \\
& & \sum_{i:(i,k)\in A} x_{ik} - \sum_{j:(k,j)\in A} x_{kj} & = & 0, & k \in V \setminus \{s,t\} \\
& & x_{ij} & \leq & K_{ij}, & (i,j) \in A \\
& & x_{ij} & \geq & 0, & (i,j) \in A
\end{array}
$$

$$
\begin{array}{llrcll}
\text{[Dual]} & \min & \sum_{(i,j)\in A} K_{ij}\gamma_{ij}, & & & \\
& \text{s.t.} & -\pi_i + \pi_j + \gamma_{ij} & \geq & 0, & (i,j) \in A \\
& & \pi_s - \pi_t & \geq & 1, & \\
& & \gamma_{ij} & \geq & 0, & (i,j) \in A
\end{array}
$$

Solving the maximum flow problem

Alternative 1: Enumerate all possible cuts and select smallest
But how do we then find the actual flow on each arc?
Also, a graph may have very many cuts

Alternative 2: Basic idea of flow algorithm:
Find paths with positive capacity through the network
Push as much flow as possible on these without violating the capacity constraints
Repeat until no capacity left

Administration: For each direction of an arc, keep track of the *residuals* (remaining capacities) $(\overline{K}_{ij}, \overline{K}_{ji})$ each time flow is pushed along the arc

Minimun cut

- A *cut* is a set of arcs which, when deleted, interrupt all flow in the network between the source s and the sink t
- The *cut capacity* equals the sum of capacities on all the arcs through the cut
- Finding the minimum cut is equal to solve the dual of the max flow problem
- Theorem: value of maximum flow = value of minimum cut (strong duality)

Max Flow Algorithm

Step 1: **Initialize** residuals $(c_{ij}, c_{ji}) = (\bar{K}_{ij}, \bar{K}_{ji})$, $i :=$source, goto 2

Step 2: Find S_i, set of nodes reachable from i with $c_{ij} > 0$
If $S_i = \emptyset$ goto 4. Else goto 3

Step 3: Choose node $k \in S_i$ with maximum c_{ik}
If $k = n$ goto 5. Else set $i := k$ and goto 2

Step 4: Getting stuck. **Backtrack** to previous node and goto 2

Step 5: **Breakthrough path found.** Calculate max flow along the path and update residuals

Step 6: **Solution.** Sum up flows on all breakthrough path
Find flow on each arc by considering the residuals