Transportation Logistics

Part I: The basics

Problems arising in the area of transportation can be solved by

- ... exact solution methods
- ... heuristics and metaheuristics

Exact methods

compute the proven optimal solution

Heuristic and metaheuristic methods

compute, hopefully good, approximate solutions. We do not know if the result is optimal. These methods are mainly used for problems that are known to be **NP-hard**.

The selection of the type of method depends on

- ... available software
- ... cost-benefit considerations
- ... complexity of the problem

Complexity

$O(1)$	$O(n)$	$O(n^c) c > 1$	$O(c^n) c > 1$
constant	linear	polynomial	exponential
determine if number is even or odd	find max in an array	maximum matching for bipartite graphs	solving the TSP with dynamic programming

run time of Simplex for LPs (avg. case) is polynomial: linear # of iterations wrt constraints; each iteration approx. a quadratic # number of evaluations.

en.wikipedia.org/wiki/Big_O_notation

Exact solution methods

- Linear programs (LP): Simplex algorithm
- Integer programs (IP): branch and bound, branch and cut, branch and price...
- Mixed integer programs (MIP): branch and bound, branch and cut, branch and price...

For many problems that can be formulated as IPs or MIPs, so far, no polynomial time algorithms are known. This is true for most vehicle routing problems.

For some problems the integrity requirements are fulfilled automatically (Transportation Problem, Assignment Problem).

In some cases, the optimization problem can be formulated as an IP or MIP but there exist algorithms that solve them in polynomial time.

Heuristics

- **Construction heuristics**: generate a feasible solution from scratch
- **Improvement heuristics**: improve an existing feasible solution
- **Combinations** of both methods
- **Metaheuristics**: generic methods, applicable to a wide range of NP-hard problems

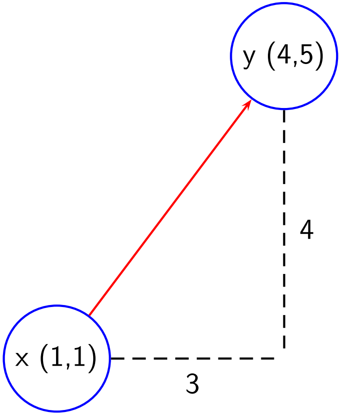
Construction heuristics and improvement heuristics are usually tailored to the problem; metaheuristics apply general concepts; within metaheuristics, construction and improvement heuristics may be applied.

Decisions in many routing and transportation problems are made on the basis of costs c_{ij} and distances d_{ij}

The most common distance measures:

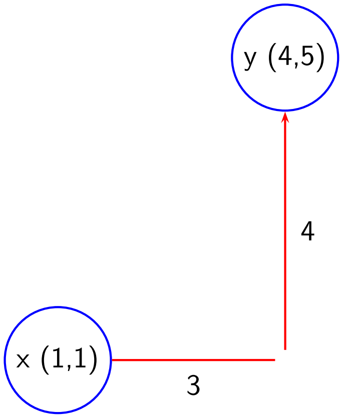
- Euclidean distance
- Manhattan distance
- Maximum distance
- Distance based on street network

Euclidean Distance



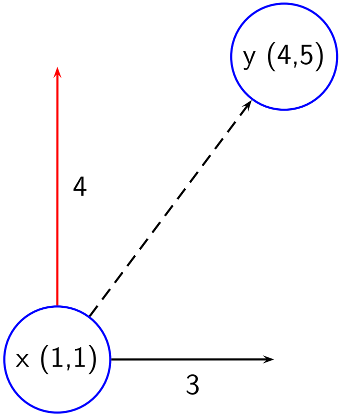
$$d(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Manhattan Distance



$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

Maximum Distance



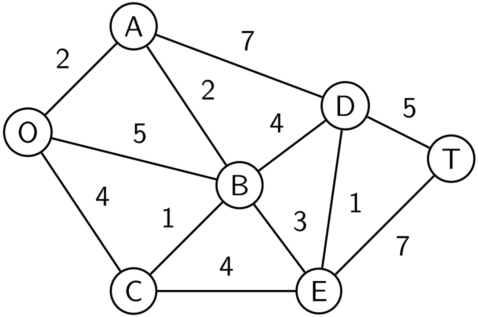
$$d(x,y) = \max_{i=1\ldots n} |x_i - y_i|$$

movement of cranes or plotter pens

Many vehicle routing and transportation problems are defined on graphs.

The weight of an arc or edge is used to represent cost, time or distance.

The nodes or vertices of the graph represent facilities, customers, depots, hubs, ...



Our graph represents a natural park and each node represents a view point. The view points are connected by hiking trails which are represented by the edges.

Example 1

Problem situation

Telephone cables are to be installed in such a way that between all view points a telephone connection is possible.
Where should the telephone cables (possible positions are the edges) be positioned such that all stations are connected to the telephone network and the total length of all the cables employed is as short as possible?

Optimization problem

Minimum spanning tree.

Example 2

Problem situation

A group of hikers would like to know what's the shortest path from the entrance O to the viewpoint at T.

Optimization problem

Shortest path.

Example 3

Problem situation

To preserve the wild life, each trail may only be traversed by a given number of people per day (given by the weight of the edge).
All hikers enter the park at the entrance O.
How many hikers may at most reach the view point at T per day?

Optimization problem

Maximum flow.

Example 4

Problem situation

At the end of the day one of the scouts has to check all viewpoints. In which order should the view points be visited such that the total travel time is minimized?

Optimization problem

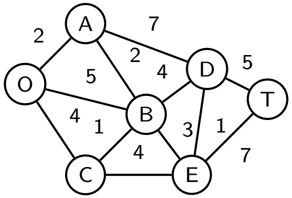
Traveling salesman/salesperson problem (TSP).

Definitions

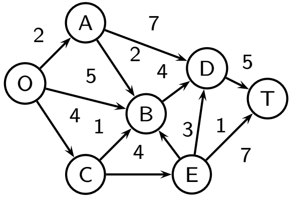
Graph

A graph consists of several nodes/vertices (more than 1) which are connected by edges/arcs.

an undirected graph (only edges)



a directed graph (digraph) (only arcs)



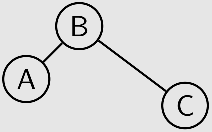
a mixed graph contains arcs and edges.

a complete graph: each pair of vertices is connected by an edge.

Definitions

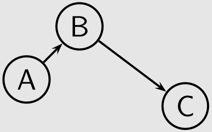
Chain

A chain consists of several arcs which connect to nodes.



Path

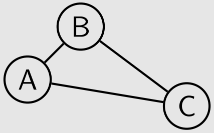
Like a chain but oriented.



Definitions

Cycle

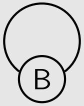
A cycle connects a node with itself without using an edge twice.



Loop

A loop connects an node with itself.

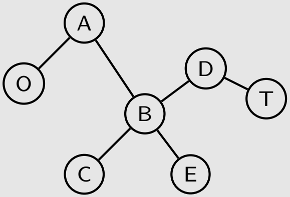
A simple graph is loopless.



Definitions

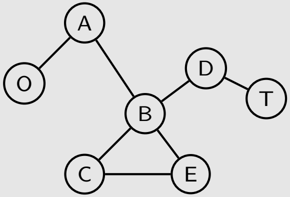
Tree

A tree is a connected graph that does not contain cycles.
A graph consisting of n nodes is connected if it contains $(n-1)$ edges and no cycles.

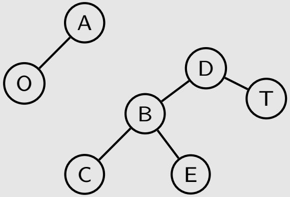


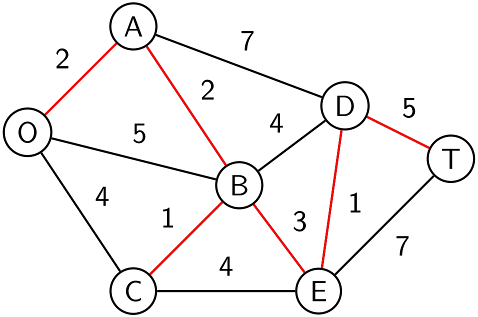
Definitions

Not a tree because of cycle



Not a tree because not connected





Since our graph consists of $n = 7$ nodes our minimum spanning tree has to consists of $(n-1)$= 6 edges and it must not contain cycles. It has a weight of 14.

Kruskal's algorithm

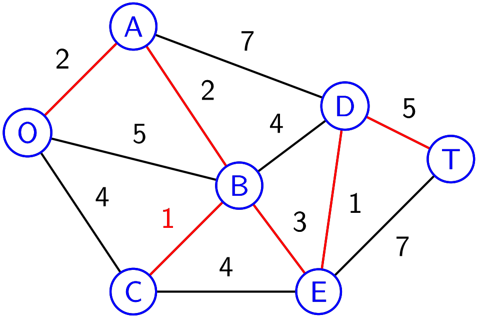
1. Find the shortest edge in the graph. This edge becomes the first edge of the MST.
2. **Repeat** until all nodes are connected
   - Find the shortest edge that connects a not yet connected node to a connected one.
   - Add this edge to the MST.

In the case of ambiguity (more than one edge could be chosen), the selection can be done arbitrarily.

```
MST( $G$, $n$ )
  sort the edges of $G$ in ascending order in queue $Q_{edges}$
  while $|E_{MST}| < (n-1)$ do
      pop the first edge $e$ from $Q_{edges}$
      if $E_{MST} \cup \{e\}$ does not lead to a cycle then
          $E_{MST} = E_{MST} \cup \{e\}$
      end if
  end while
```

Complexity: $O(|E| \log |V|)$ $V ... setofvertices$, $E ... setofedges$

Kruskal's algorithm - Example



Shortest path problem

Determine the shortest path in a graph from a source node O to a sink node T.

Shortest path algorithms can be classified as follows:

- Tree algorithms: determine the shortest path between two nodes (Dijkstra, Bellman).
- Methods that determine the shortest path from all nodes to all other nodes (distance matrix!). (Triple algorithm)

A shortest path algorithm

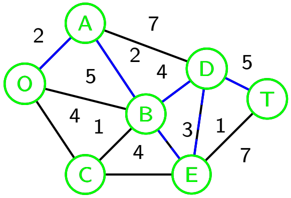
Source: Hillier, Lieberman (1995) 'Introduction to Operations Research'

- set the origin node (O) to solved; $n = 1$
- **Repeat until the destination node T is reached**
  1. find nth nearest node to the origin. The nth nearest node can be any node that is directly connected to a solved node.
  2. $n := n + 1$

For undirected, connected graphs without negative distances.

A shortest path algorithm - Example

n	solved nodes directly connected to unsolved nodes	closest connected unsolved node	total distance involved	nth nearest node	min dist.	last connection
1	O	A	2	A	2	OA
2,3	O	C	4	C	4	OC
	A	B	$2+2=4$	B	4	AB
4	A	D	$2+7=9$			
	B	E	$4+3=7$	E	7	BE
	C	E	$4+4=8$			
5	A	D	$2+7=9$			
	B	D	$4+4=8$	D	8	BD
	E	D	$7+1=8$	D	8	ED
6	D	T	$8+5=13$	T	13	DT
	E	T	$7+7=14$			



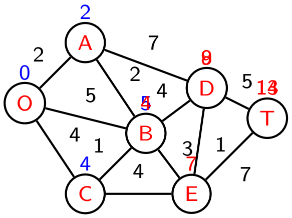
Dijkstra's algorithm

1. **Initialization** $(n = 0)$
   The tentative shortest distance of each node i from the origin $D[i]$ is set to the direct distance d_{0i} and node 0 is set to visited. The tentative direct predecessor of each i is set to the source $V[i] = O$.
2. **Repeat** until T is visited /all nodes are visited $(n > 0)$
   - the current node i is the unvisited node with the minimal distance from O, given by $D[i]$. It is set to visited. This node is the n-th next node to the source O. The shortest distance to O is $D[i]$ and its direct predecessor is $V[i]$.
   - Determine all unvisited nodes j, which are reachable from i via a direct edge. **if** $D[i] + d_{ij} < D[j]$ **then** the path via i to j is shorter than the so far known shortest path to j and we set $D[j] = D[i] + d_{ij}$ and $V[j] = i$.

$$d_{ij} = \begin{cases} \text{length of the edge between } i \text{ and } j \text{, if edge exists.} \\ \infty \text{, otherwise.} \end{cases}$$

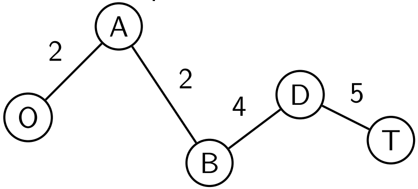
Dijkstra's Algorithm - Example

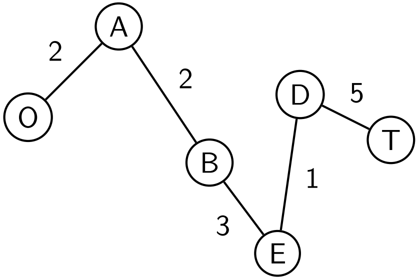
n	$D[O]$, $V[O]$	$D[A]$, $V[A]$	$D[B]$, $V[B]$	$D[C]$, $V[C]$	$D[D]$, $V[D]$	$D[E]$, $V[E]$	$D[T]$, $V[T]$	visited
0	0, O	2, O	5, O	4, O	∞, O	∞, O	∞, O	O, 0
1		2, O	5, O	4, O	∞, O	∞, O	∞, O	A, 2
2			4, A	4, O	9, A	∞, O	∞, O	B, 4
3				4, O	8, B	7, B	∞, O	C, 4
4					8, B	7, B	∞, O	E, 7
5					8, B,E		14, E	D, 8
6							13, D	T, 13



Dijkstra's Algorithm - an Example

Two shortest paths.





Bellman(-Ford) algorithm

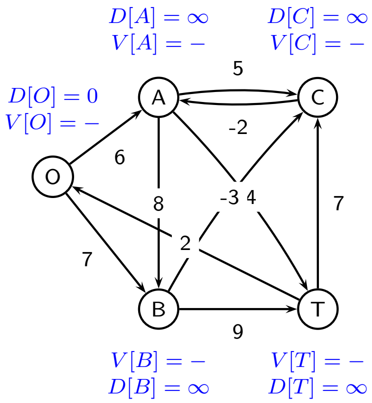
Bellman invented dynamic programming. The Bellman(-Ford) algorithm for shortest path problems is sometimes referred to as label correcting algorithm.

```
1 Initialization set $D[O] = 0$, for all other nodes $i$ set $D[i] = \infty$
2 Repeat $|V| - 1$ times
    • Repeat for each edge $(i, j)$
        • if $D[j] > D[i] + d_{ij}$ then
          $D[j] = D[i] + d_{ij}, V[j] = i$
3 Repeat for each edge $(i, j)$
    • if $D[i] + d_{ij} < D[j]$ then
      Error: Graph contains a negative weight cycle.
```

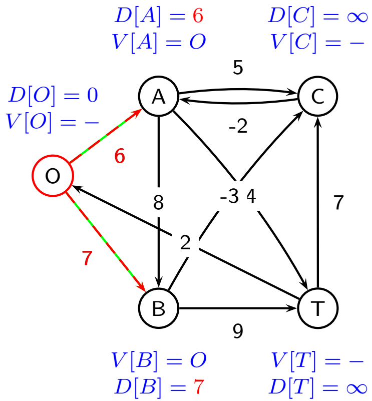
Complexity: $O(|V||E|)$ (V...set of vertices; E...set of edges)
For directed graphs. Edge weights/distances may be negative. (Dijkstra's algorithm cannot be applied in this case)

Source: Corman et al. (2001) Introduction to algorithms, 2nd edition

Bellman(-Ford) algorithm - Example



Initialization

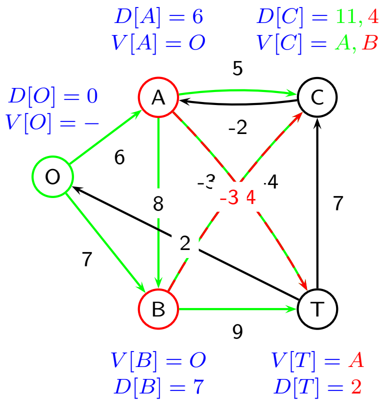


Iteration 1

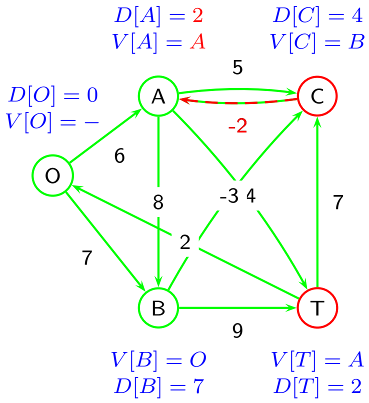
$d_{OA} = 6,\ d_{OB} = 7,\ d_{AB} = 8,\ d_{AC} = 5,\ d_{AT} = -4,\ d_{BC} = -3,\ d_{CA} = -2,\ d_{TO} = 2,\ d_{TC} = 7$

(we iterate over the edges in this order)

Bellman(-Ford) algorithm - Example



Iteration 2

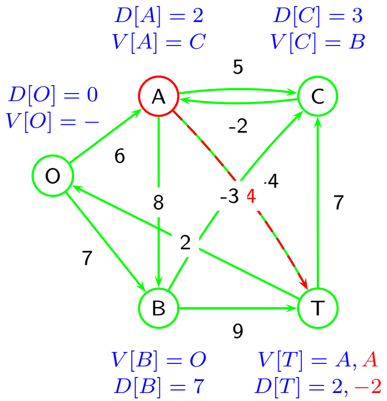


Iteration 3

$d_{OA} = 6$, $d_{OB} = 7$, $d_{AB} = 8$, $d_{AC} = 5$, $d_{AT} = -4$, $d_{BC} = -3$, $d_{CA} = -2$, $d_{TO} = 2$, $d_{TC} = 7$

(we iterate over the edges in this order)

Bellman(-Ford) algorithm - Example



Iteration 4

$d_{OA} = 6$, $d_{OB} = 7$, $d_{AB} = 8$, $d_{AC} = 5$, $d_{AT} = -4$, $d_{BC} = -3$, $d_{CA} = -2$, $d_{TO} = 2$, $d_{TC} = 7$

(we iterate over the edges in this order)

Shortest paths between all nodes in the network

We want to determine the shortest paths between every pair of nodes in our graph.

Two solutions:

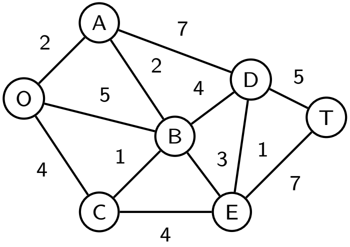
1. use a shortest path algorithm and use every node as the source node in turn.
2. BETTER: apply the Triple algorithm.

The Triple Algorithm

k_{ij}...shortest distance between i and j
v_{ij}...direct predecessor of j

1. **Initialization**
   set $k_{ij} = d_{ij}$; set $v_{ij} = i$
2. **Repeat** for each vertex $n \in V$
   1. for all i and j
      - **if** $k_{in} + k_{nj} < k_{ij}$ **then** the new path via n is shorter than the so far known shortest path from i to j and we se $k_{ij} = k_{in} + k_{nj}$

The Triple algorithm - Example



The k_{ij} matrix.

	O	**A**	**B**	**C**	**D**	**E**	**T**
O	0	2	5	4	∞	∞	∞
A	2	0	2	∞	7	∞	∞
B	5	2	0	1	4	3	∞
C	4	∞	1	0	∞	4	∞
D	∞	7	4	∞	0	1	5
E	∞	∞	3	4	1	0	7
T	∞	∞	∞	∞	5	7	0

Initialization

The Triple algorithm - Example

The k_{ij} matrix.

	O	**A**	**B**	**C**	**D**	**E**	**T**
O	0	2	5	4	∞	∞	∞
A	2	0	2	6	7	∞	∞
B	5	2	0	1	4	3	∞
C	4	6	1	0	∞	4	∞
D	∞	7	4	∞	0	1	5
E	∞	∞	3	4	1	0	7
T	∞	∞	∞	∞	5	7	0

Paths via O
e.g., from A to C: previously no connection; now A-O-C: $2 + 4 = 6$

The k_{ij} matrix.

	O	**A**	**B**	**C**	**D**	**E**	**T**
O	0	2	4	4	9	∞	∞
A	2	0	2	6	7	∞	∞
B	4	2	0	1	4	3	∞
C	4	6	1	0	13	4	∞
D	9	7	4	13	0	1	5
E	∞	∞	3	4	1	0	7
T	∞	∞	∞	∞	5	7	0

Paths via A
e.g., from O to B: previous distance 5; now O-A-B: $2 + 2 = 4$

The Triple algorithm - Example

The k_{ij} matrix.

	O	A	B	C	D	E	T
O	0	2	4	4	8	7	∞
A	2	0	2	3	6	5	∞
B	4	2	0	1	4	3	∞
C	4	3	1	0	5	4	∞
D	8	6	4	5	0	1	5
E	7	5	3	4	1	0	7
T	∞	∞	∞	∞	5	7	0

Paths via B

The k_{ij} matrix.

	O	A	B	C	D	E	T
O	0	2	4	4	8	7	∞
A	2	0	2	3	6	5	∞
B	4	2	0	1	4	3	∞
C	4	3	1	0	5	4	∞
D	8	6	4	5	0	1	5
E	7	5	3	4	1	0	7
T	∞	∞	∞	∞	5	7	0

Paths via C

The Triple algorithm - Example

The k_{ij} matrix.

	O	**A**	**B**	**C**	**D**	**E**	**T**
O	0	2	4	4	8	7	13
A	2	0	2	3	6	5	11
B	4	2	0	1	4	3	9
C	4	3	1	0	5	4	10
D	8	6	4	5	0	1	5
E	7	5	3	4	1	0	6
T	13	11	9	10	5	6	0

Paths via D

The k_{ij} matrix.

	O	**A**	**B**	**C**	**D**	**E**	**T**
O	0	2	4	4	8	7	13
A	2	0	2	3	6	5	11
B	4	2	0	1	4	3	9
C	4	3	1	0	5	4	10
D	8	6	4	5	0	1	5
E	7	5	3	4	1	0	6
T	13	11	9	10	5	6	0

Paths via E

The Triple algorithm - Example

The k_{ij} matrix.

	O	**A**	**B**	**C**	**D**	**E**	**T**
O	0	2	4	4	8	7	13
A	2	0	2	3	6	5	11
B	4	2	0	1	4	3	9
C	4	3	1	0	5	4	10
D	8	6	4	5	0	1	5
E	7	5	3	4	1	0	6
T	13	11	9	10	5	6	0

No further changes.
Distance matrix complete.

The first row of the matrix corresponds to the solutions of our tree algorithms!

Paths via E

The maximum flow problem

Determine the maximum flow from a source to a sink node in a given graph.

In this case, the weights of the edges correspond to capacities, i.e. the maximum flow on this edge.

Ford-Fulkerson Algorithm

a.k.a. the 'Augmenting path algorithm'

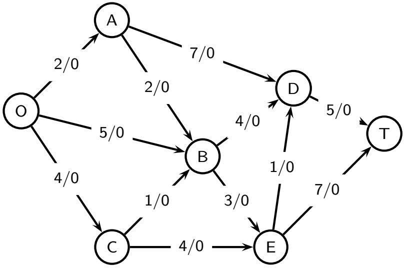
1. initialize the flow f to 0
2. **Repeat** while there exists an augmenting path p from O to T
   - augment the flow f along path p: identify the residual capacity c^* of the path p and decrease by c^* the residual capacity on each arc of the path p and increase the residual capacity of each arc in the opposite direction on the path p.

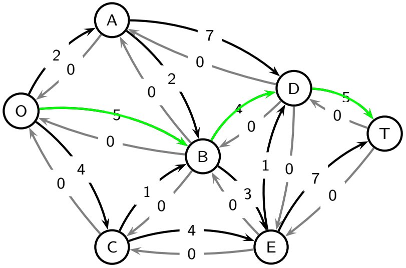
An augmenting path is a directed path from O to T in the residual network such that every arc on the path has strictly positive residual capacity.

Source: Corman et al. (2001) Introduction to algorithms, 2nd edition

Hillier, Lieberman (1995) 'Introduction to Operations Research'

Ford-Fulkerson Algorithm - Example

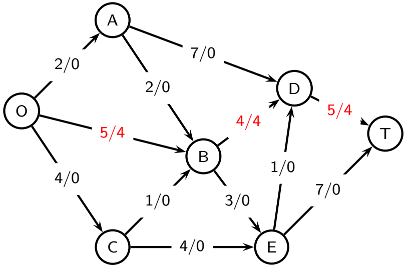




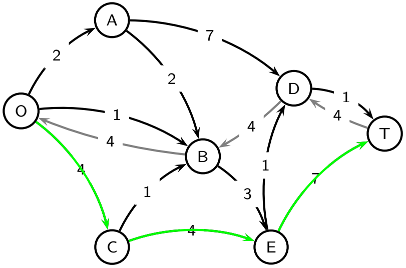
The flow network **The residual network**

The first augmenting path $p = O - B - D - T$ and $c^* = 4 \rightarrow f = 4$

Ford-Fulkerson Algorithm - Example



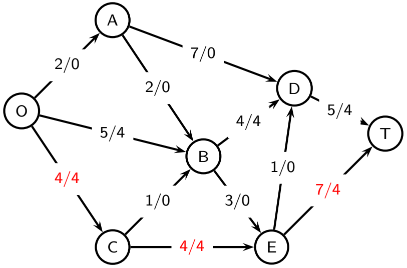
The flow network



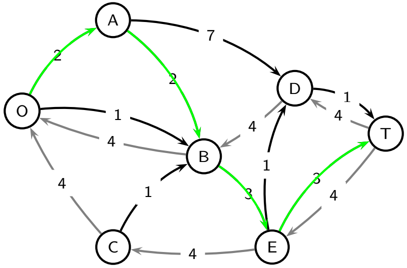
The residual network

The second augmenting path $p = O - C - E - T$ and $c^* = 4 \rightarrow f = 8$

Ford-Fulkerson Algorithm - Example



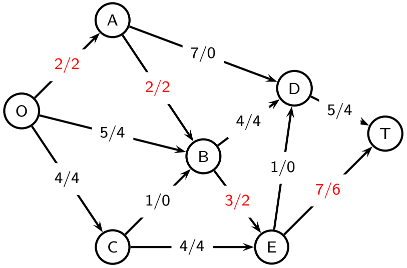
The flow network

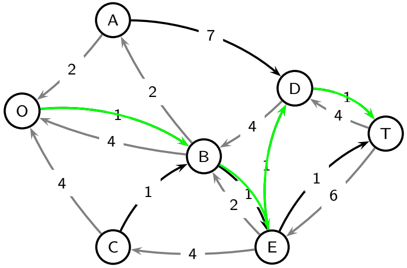


The residual network

The third augmenting path $p = O - A - B - E - T$ and $c^* = 2 \rightarrow f = 10$

Ford-Fulkerson Algorithm - Example

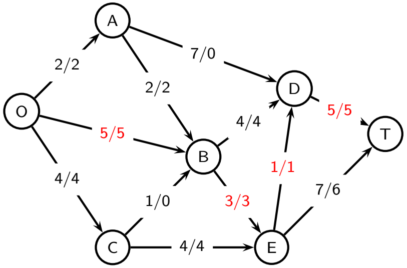




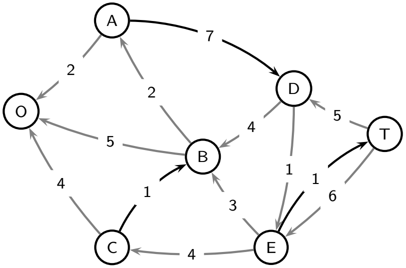
The flow network **The residual network**

The fourth augmenting path $p = O - B - E - D - T$ and $c^* = 1 \rightarrow f = 11$

Ford-Fulkerson Algorithm - Example



The flow network



The residual network

No additional augmenting path; max flow $f = 11$

Max-flow min-cut theorem

Cut

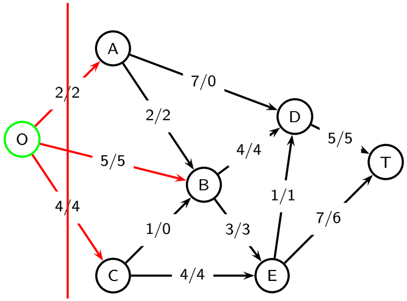
any set of directed arcs containing at least one arc from every directed path from O (supply) to T (demand node).

Cut value

sum of the capacities of the arcs (in the specified direction) of the cut.

Max-flow min-cut theorem

for any network with a single supply and a single demand node, the maximum feasible flow is equal to the minimum cut value across all cuts of the network.



max-flow = min-cut = 11

References

T.H. Corman, C.E. Leiserson, R.L. Rivest, C. Stein (2001) Introduction to algorithms. MIT Press ISBN-10:0-262-03293-72 (2nd Editon; 3rd Edition appeared in 2009)

E. W. Dijkstra (1959) A note on two problems in connexion with graphs. NUMERISCHE MATHEMATIK 1 (1): 269-271, DOI: 10.1007/BF01386390 `http://www.springerlink.com/content/uu8608u0u27k7256`

F. Hillier, G. Lieberman (1995) Introduction to Operations Research, McGraw-Hill. (In German published by Oldenbourg)