Maximum Flows

Originated in the 50s to study rail networks.

Definitions:

A <u>flow network</u> is a directed graph $G=(V,E)$ where each edge has a <u>capacity</u> $c(e) \in \mathbb{R}^+$, along with a designated <u>source</u> $s \in V$ and <u>sink</u> $t \in V$.

For convience, write $c(e)=0$ if $e \notin E$, no edges enter s and no edges leave t.

A <u>flow</u> is a function $f: E \to \mathbb{R}^+$ satisfying the following constraints:

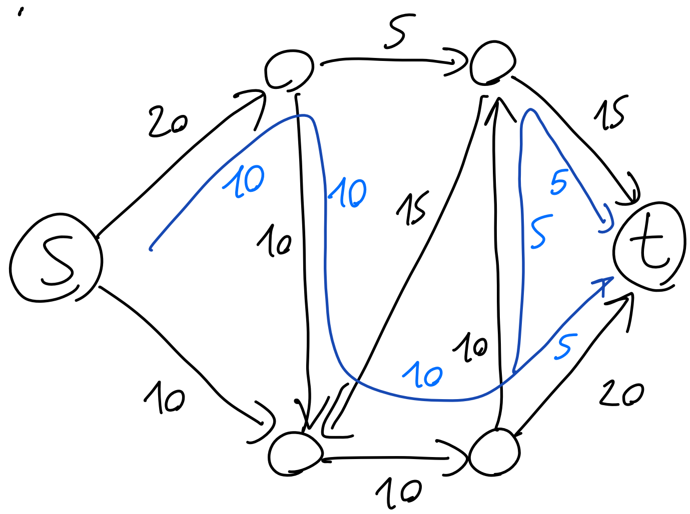
1) (Capacity) $\forall e \in E \quad f(e) \leq c(e)$
2) (Conservation) $\forall u \in V \setminus \{s,t\}$ we have

$$\sum_{\substack{v \in V \\ \text{s.t. } (v,u) \in E}} f(v,u) = \sum_{\substack{v \in V \\ \text{s.t. } (u,v) \in E}} f(u,v)$$

The <u>value</u> of a flow is $|f| = \sum_{\substack{v \in V \\ \text{s.t. } (s,v) \in E}} f(s,v)$

Maximum flow problem: given a flow network find a flow f of maximum value.

Example:



a flow of value 10

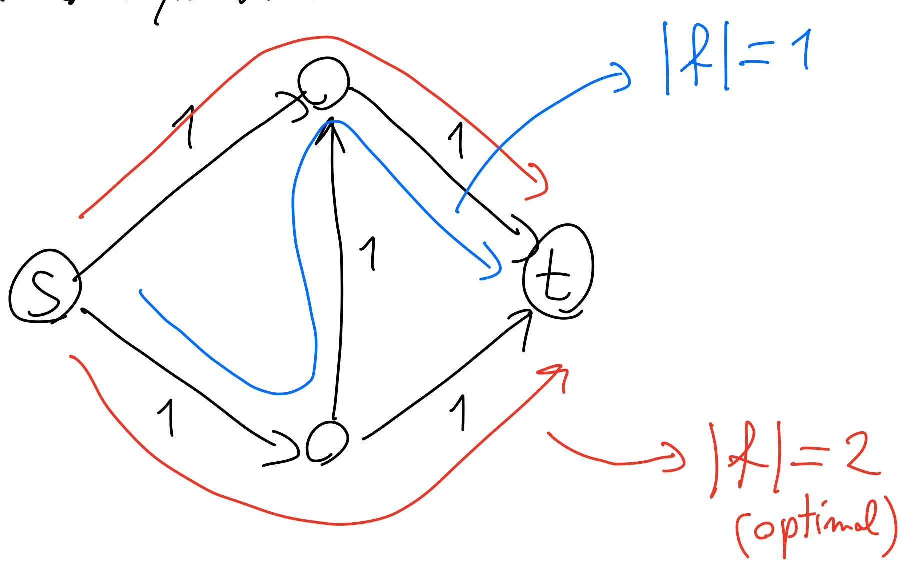
Applications: rail/airline/road network, electrical networks, communication network, liquid transportation networks; moreover, it can be applied to solve several other problems in comp. science (e.g. bipartite matching)

Max flow reduces to linear programming (like many other problems) but there are more efficient special-purpose algorithms. We'll see one (Ford-Fulkerson) but there are plenty more efficient algorithms → see Further Reading.

A natural idea: be greedy:

- find a path from s to t (in linear time using BFS)
- send as much flow along it as possible
- Update capacities
- remove edges that have 0 remaining capacity
- repeat until the graph has no s-t paths

This won't always work:



Idea: revise/undo some of this flow later in the algorithm; how? By "pushing back" some flow through new edges in the reverse direction.

Definition: given a flow network G and a flow f, the <u>residual network</u> of G w.r.to flow f, G_f, is a network with vertex set V and with edge set E_f as follows:

for every edge $e=(u,v)$ in G

- if $f(e) < c(e)$, add e to G_f with capacity $c_f(e) = c(e) - f(e)$
- if $f(e) > 0$, add another edge (v,u) to G_f with capacity $c_f(e) = f(e)$

The Ford-Fulkerson algorithm repeatedly finds an s-t path P in G_f (e.g., using BFS) and uses P to increase the current flow. P is called <u>augmenting path</u>.

Ford-Fulkerson (G, s, t) 1956

initialize $f(e) = 0$ for all $e \in G.E$

$G_f = G$

while there exists an augmenting path P in G_f do

  let $\Delta_p = \min_{e \in P} c_f(e)$ // Δ_p is the "bottleneck" capacity in P

  for each edge $e = (u,v) \in P$ do

    if $(u,v) \in G.E$ then

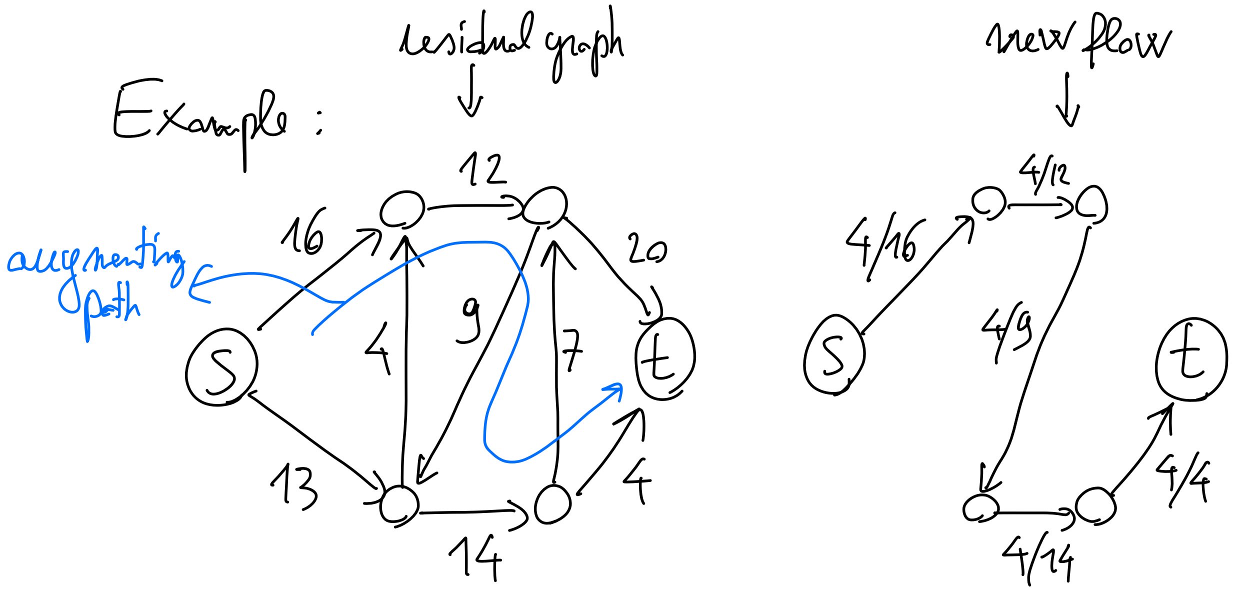
      $f(u,v) = f(u,v) + \Delta_p$

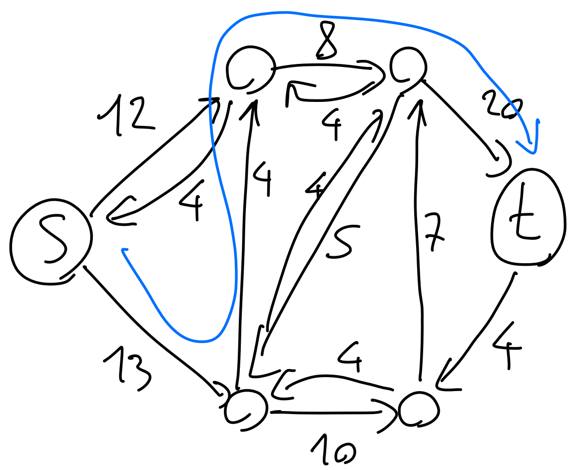
    else $f(v,u) = f(v,u) - \Delta_p$

  update the residual graph G_f

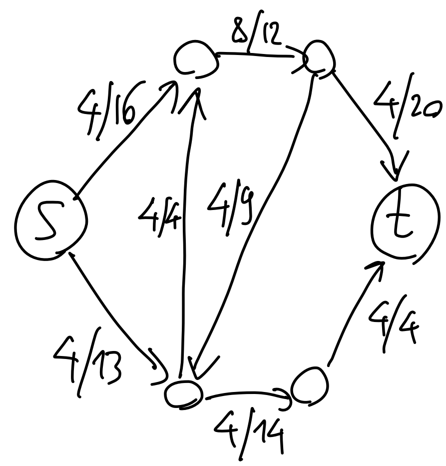
return f

Example:

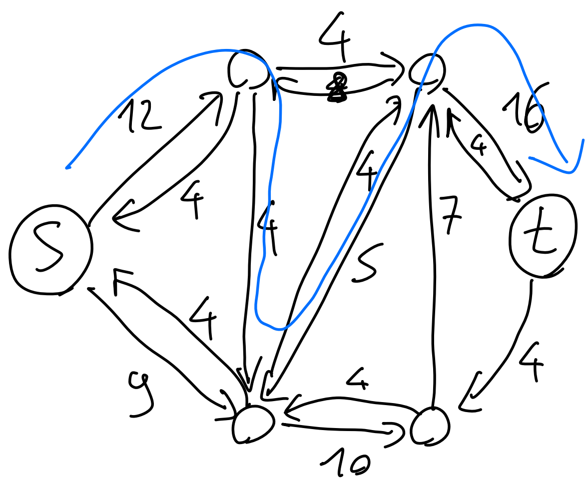




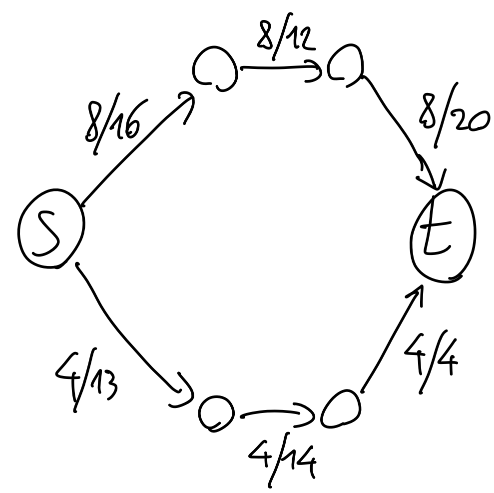
S
t
12
4
8
4
20
4
4
5
7
13
4
4
10



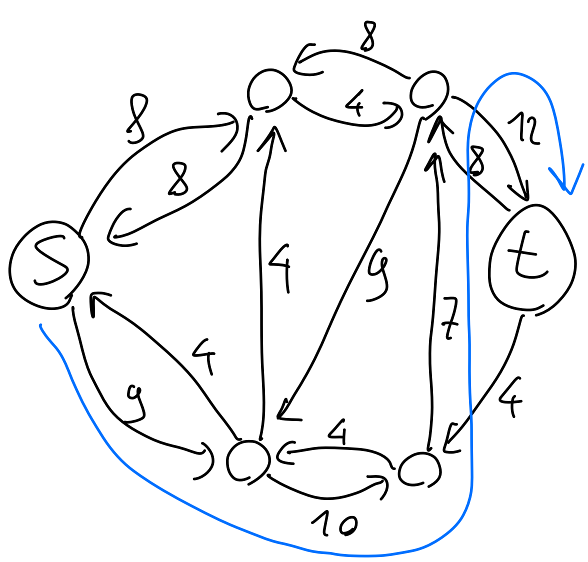
S
t
8/12
4/16
4/20
4/4
4/9
4/13
4/4
4/14



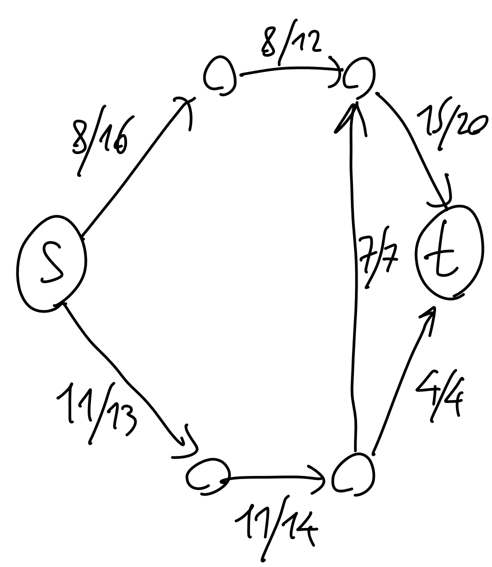
S
t
4
12
4
4
16
4
4
5
7
4
4
9
4
10



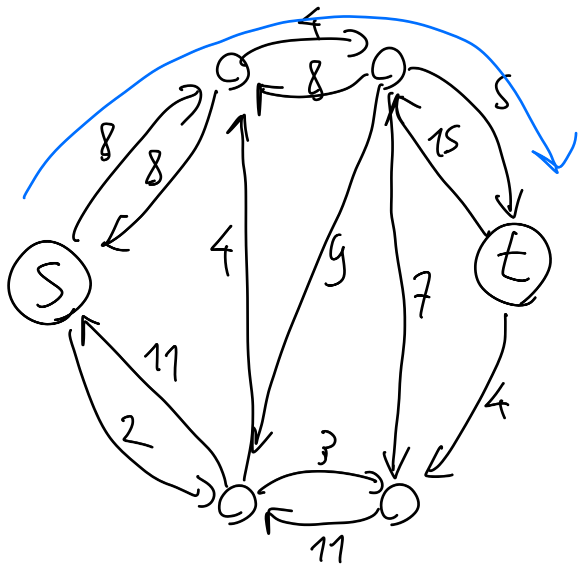
S
t
8/12
8/16
8/20
4/13
4/4
4/14

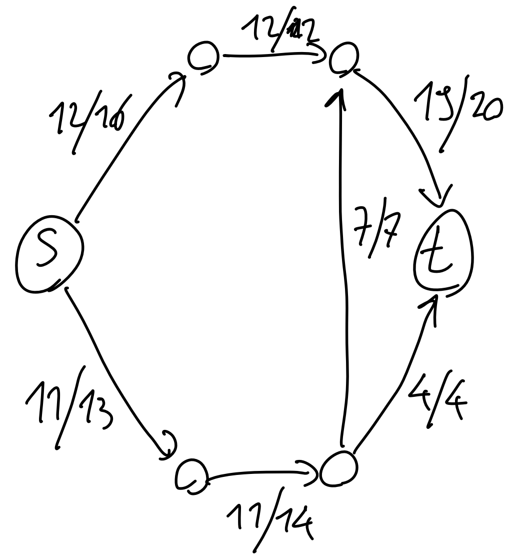


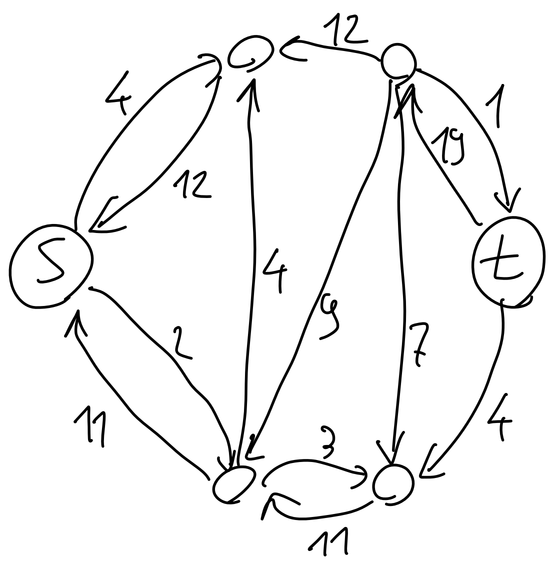
S
t
8
8
8
4
12
8
4
9
7
4
9
4
4
10



S
t
8/12
8/16
15/20
7/7
11/13
4/4
11/14







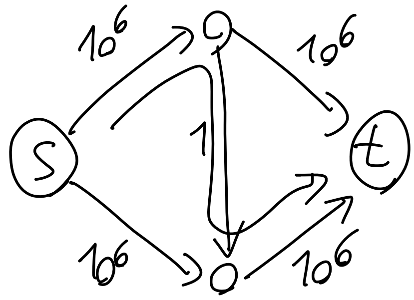
no augmenting path $\Rightarrow$ is a max flow, of value 23

Complexity: assume capacities are integers; then

- the flow value increases by ≥ 1 in each iteration
- the complexity of each iteration is $O(m)$

$\Rightarrow O(m \cdot |f^*|)$ $\quad$ f^* is a max flow

A flow network for which F-F can take $\Theta(m \cdot |f^*|)$ time



input size: $O(m \log U)$ U = max capacity

$O(m|f^*|) = O(mnU)$ "pseudo-polynomial"