Network Flows
Linear Programming

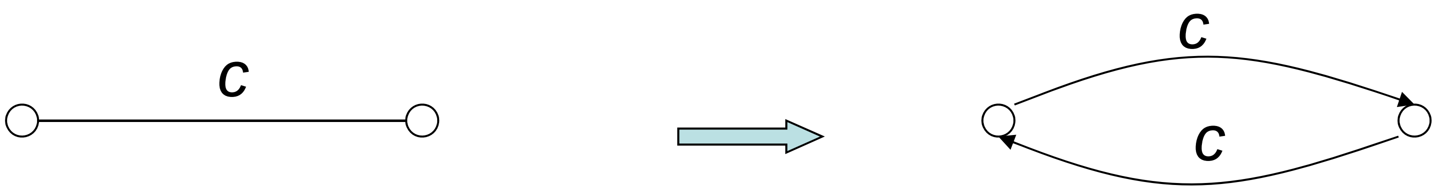
CS 4231, Fall 2017

Mihalis Yannakakis

Flow Network

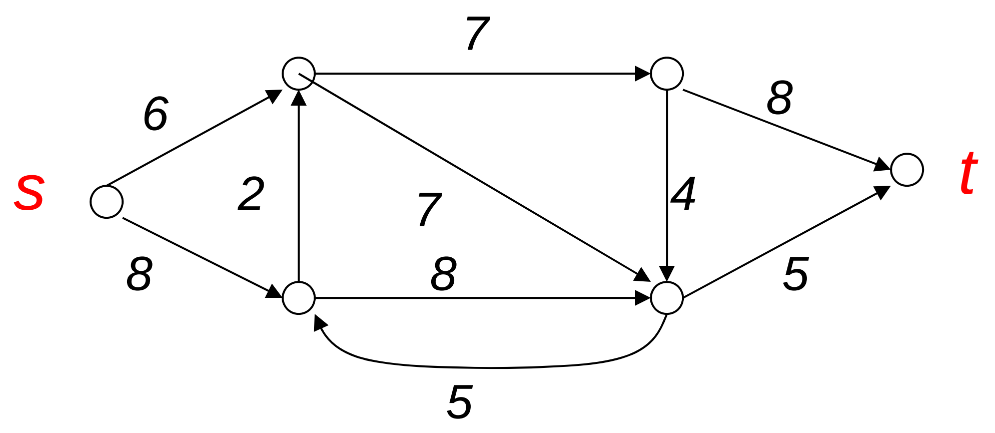
- Directed Graph G=(N,E)
- Positive capacities on edges c: $E \to R_{>0}$
  - 0 capacity ⇔ missing edge
- Source node s, Sink node t

- Undirected graphs: replace each undirected edge by two opposite directed edges with same capacity



Max s-t Flow Problem

Given flow network:



- Send maximum flow from s to t

Max s-t Flow problem

- Edge flow formulation

Edge flow function, $f: E \to R_{\geq 0}$ satisfies:

- Nonnegativity: $f(u,v) \geq 0, \ \forall (u,v) \in E$
- Capacity constraints : $f(u,v) \leq c(u,v)\ , \ \forall (u,v) \in E$
- Flow conservation constraints:

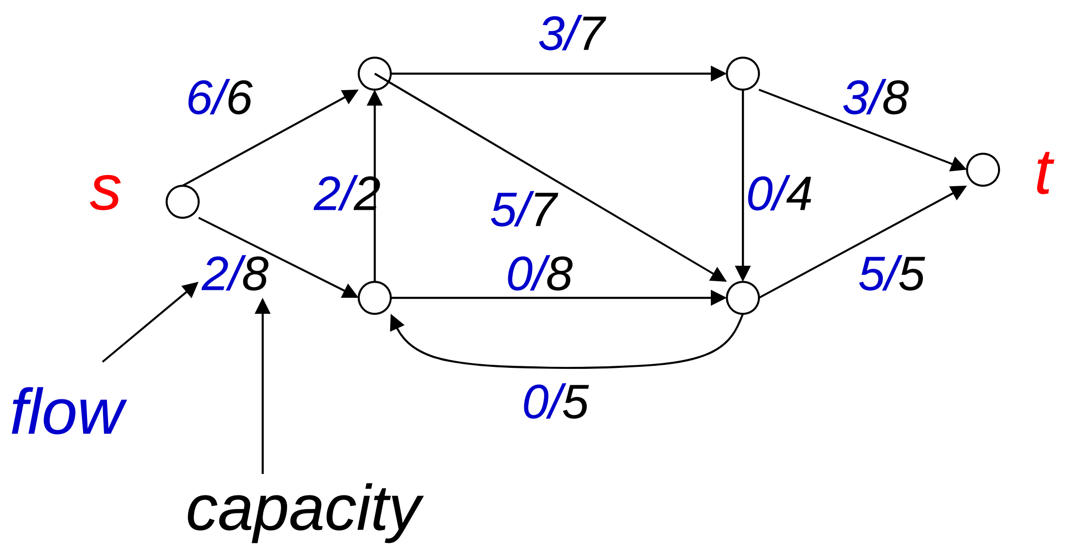
$$\sum_{v:(u,v)\in E} f(u,v) - \sum_{v:(v,u)\in E} f(v,u) = 0, \ \forall u \in N - \{s,t\}$$

Maximize value of the flow : $|f| = \sum_{(s,v)\in E} f(s,v) - \sum_{(v,s)\in E} f(v,s)$

Summing the flow conservation constraints $\Rightarrow$ net-out-flow(s) = net-in-flow(t)

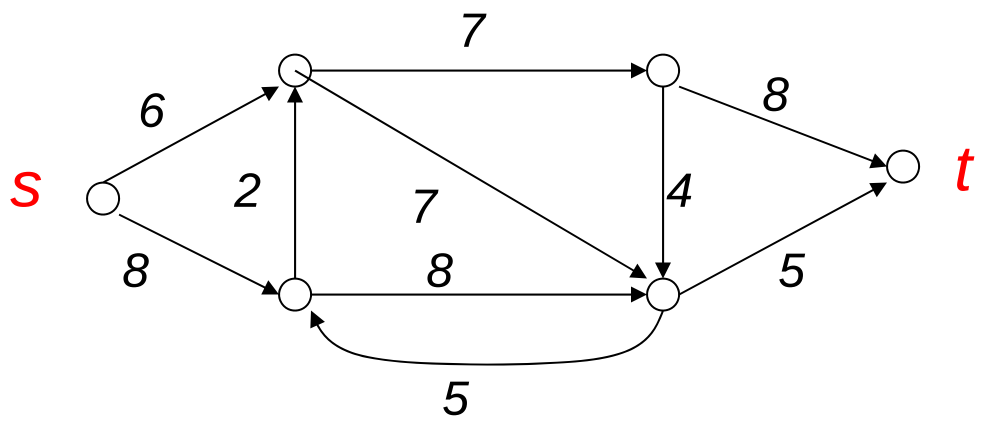
$$|f| = \sum_{(v,t)\in E} f(v,t) - \sum_{(t,v)\in E} f(t,v)$$

Example: A s-t flow



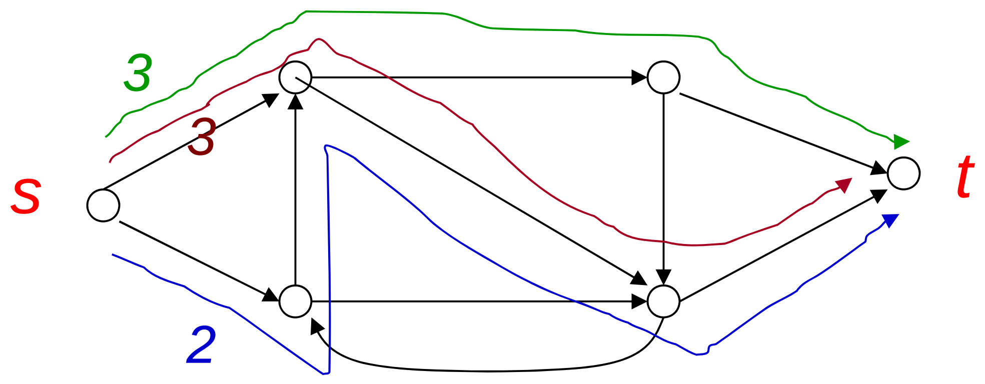
Flow value =8

Max s-t Flow Problem



Path version of flow: Pick s-t flow paths and flow amounts for each path such that total flow through each edge ≤ capacity, and sum of flows is maximized

Equivalent to the edge flow version



Decomposition of Flow into Paths

Every edge flow can be decomposed into

- A set of s-t paths, each with an associated flow, and
- A set of cycles, each with an associated flow

Proof: Consider the *support* of the flow: E'={(u,v)|f(u,v)>0}

While there is a cycle in E', take such a cycle C, associate with it the min flow of an edge on C, and subtract this amount from all edges of C, and update E' ($\rightarrow$ flow with same value)

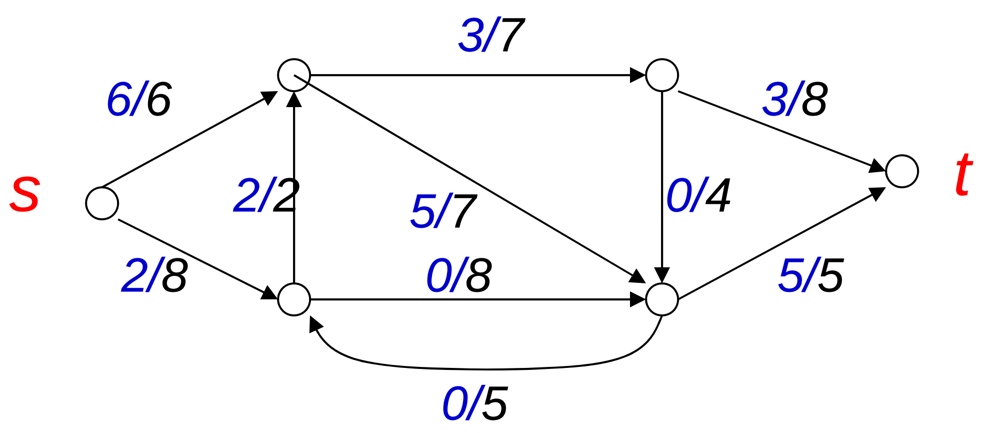
While there is a s-t path using edges in E', take such a path p, associate with it the min flow of an edge on p, and subtract this amount from all edges of p, and update E'.

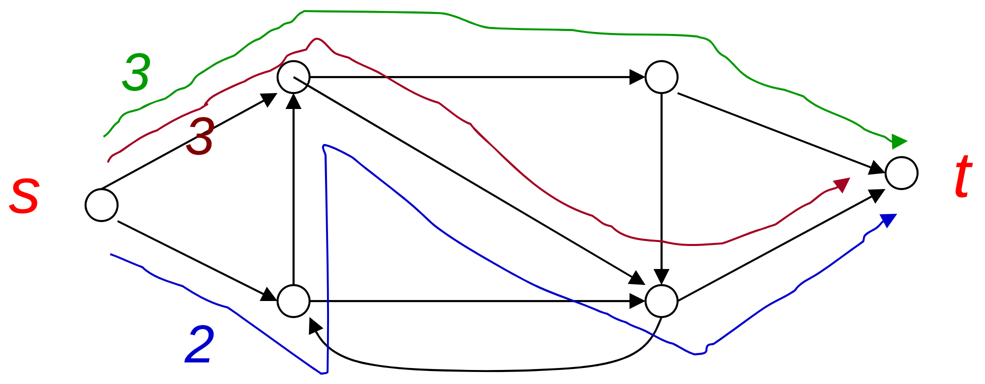
At the end, no s-t path and no cycle $\Rightarrow$ remaining flow =0

At most |E| paths and cycles

- Flow along cycles is clearly redundant and can be removed

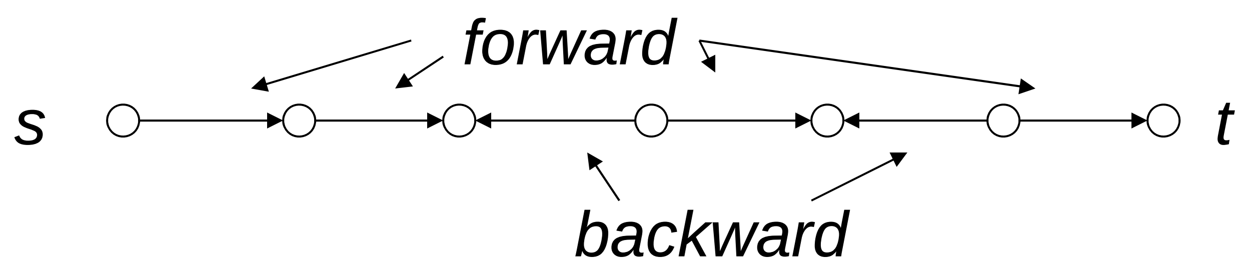
Example: Decomposition of a s-t flow





Augmenting s-t Path

- Undirected path from s to t such that forward edges of the path are not saturated (flow <capacity) and backward edges carry positive flow

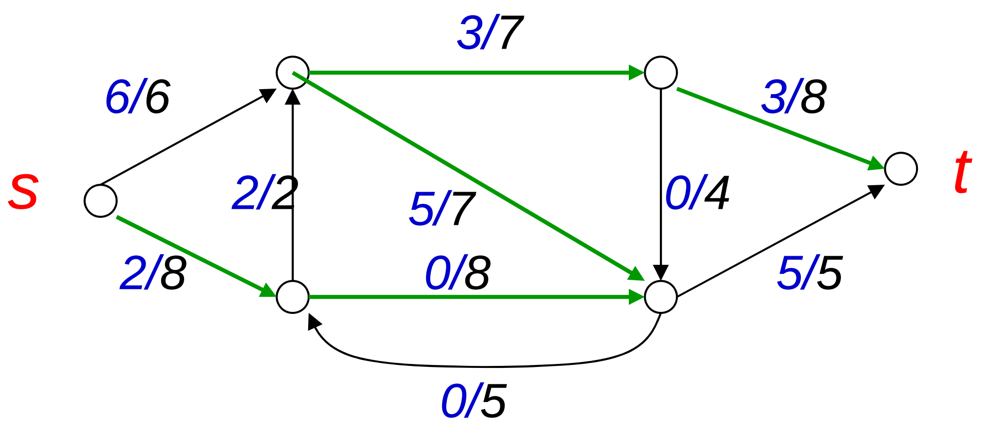


- If we increase flow in forward edges and decrease in backward edges by same amount δ then resulting flow is feasible as long as:

  - $\delta \le c(u,v)-f(u,v)$, for forward edges (u,v)
  - $\delta \le f(u,v)$, for backward edges (u,v)

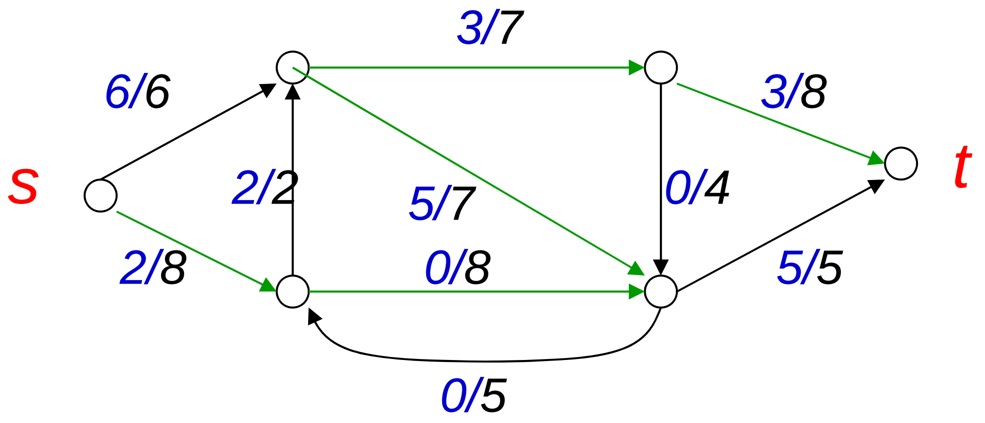
$\Big\}$ *residual capacity*

Example: Augmenting path of a s-t flow



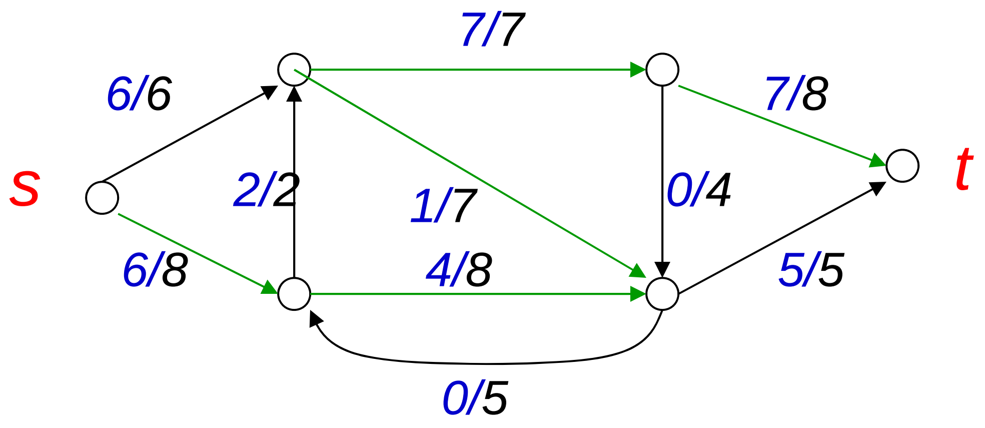
Augmenting path. *Residual capacity = 4*

Example: Augmentation of a s-t flow



Augmenting path. *Residual capacity = 4*

New flow



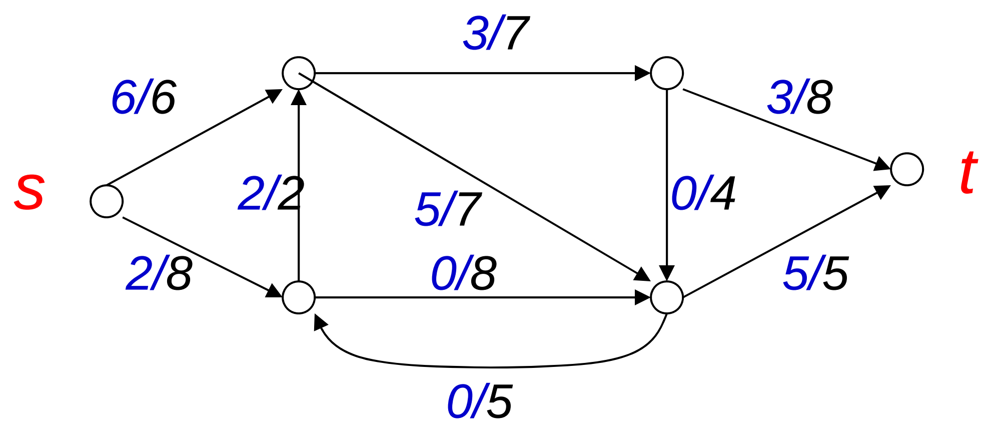
Residual Network

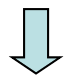
- Given network $G=(N,E,c)$ with source s, sink t, and s-t flow f define residual network $G_f=(N,E_f,c_f)$ wrt f:
- For every edge (u,v) of G,
  - If $f(u,v) < c(u,v)$ then include edge (u,v) in G_f with residual capacity $c_f(u,v)= c(u,v)-f(u,v)$
  - If $f(u,v) >0$ then include reverse edge (v,u) in G_f with residual capacity $c_f(v,u)= f(u,v)$

- Augmenting s-t paths in G wrt flow f = s-t paths in G_f
- Residual capacity of path p in $G_f = \min \{c_f(u,v) \mid (u,v) \text{ in } p \}$

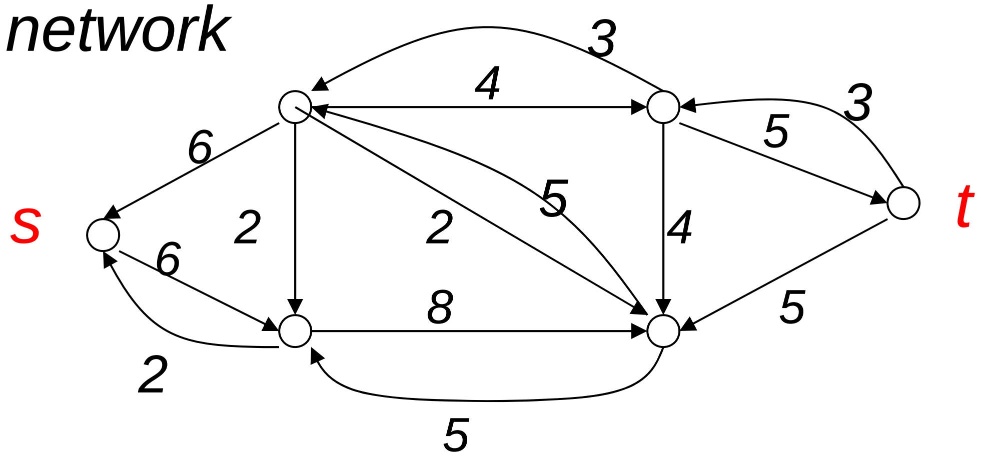
Change flow f to f' along path p: increase flow on forward edges, decrease on reverse edges

Example: A s-t flow and residual network





Residual network

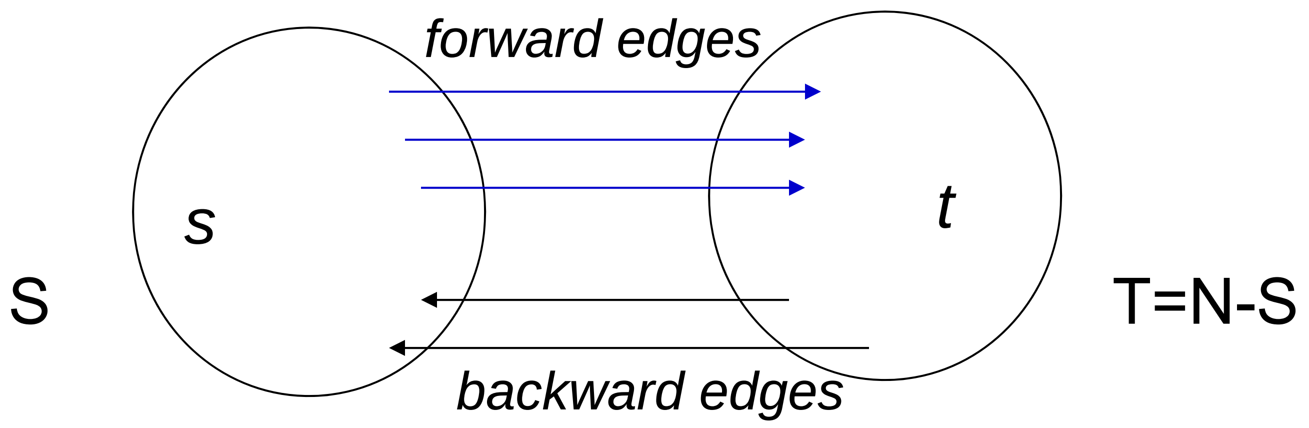


Ford-Fulkerson Algorithm

```
Initialize: For each edge $(u,v) \in E$ do  $f(u,v)=0$
            $G_f = G$
While $\exists$ path p in $G_f$ from s to t do    [augmenting path]
 { Update the flow f :
     $\delta = \min\{c_f(u,v) \mid (u,v) \in p\}$;
     for each edge (u,v) in p do  [update flow on edge or reverse]
       { if (u,v) a reverse edge ($f(v,u)>0$) then $f(v,u) = f(v,u) - \delta$
             else   $f(u,v)=f(u,v) + \delta$  }
   Update the residual graph $G_f$
 }
```

s-t Cut

- s-t Cut: A partition (S,T) of nodes such that s∈S, t ∈T
- Capacity of Cut: c(S,T)=Σ{c(u,v) : u ∈S, v ∈T}



Only forward edges count in the capacity of the cut

•Minimum cut: s-t cut of minimum capacity

Max Flow ≤ Min Cut

For any s-t flow f and any s-t cut (S,T), the value of the flow |f| is $\leq$ capacity c(S,T) of the cut

$$|f| = f(S \to T) - f(T \to S) = \sum_{\substack{(u,v)\in E \\ u\in S, v\in T}} f(u,v) - \sum_{\substack{(u,v)\in E \\ u\in T, v\in S}} f(u,v) \leq c(S,T)$$

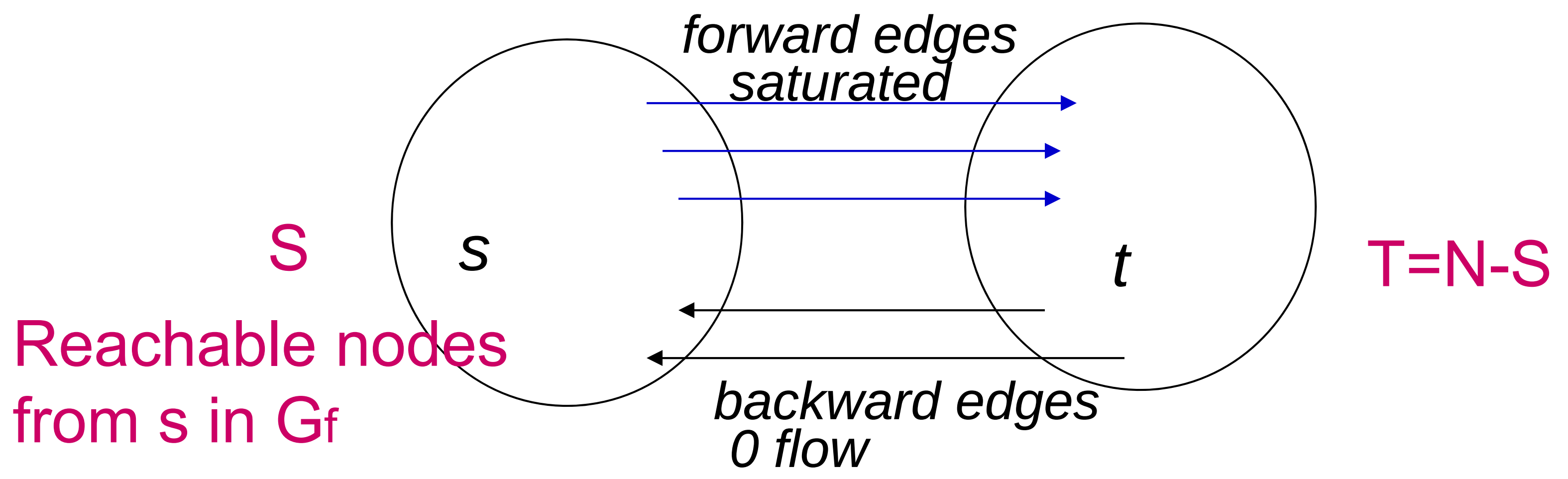
Proof: Add the flow conservation constraints for all u$\in$S-{s} and the equation $|f| = \sum_{(s,v)\in E} f(s,v) - \sum_{(v,s)\in E} f(v,s)$

The flows f(u,v), u,v $\in$S cancel, and we'll get

$$|f| = f(S \to T) - f(T \to S) = \sum_{\substack{(u,v)\in E \\ u\in S, v\in T}} f(u,v) - \sum_{\substack{(u,v)\in E \\ u\in T, v\in S}} f(u,v) \leq c(S,T)$$

Equality holds iff all forward edges of cut are *saturated* (f(u,v)=c(u,v)), and all backward edges have 0 flow.

No augmenting path ⇒ Max flow



Residual network G_f has no edges coming out of S ⇒

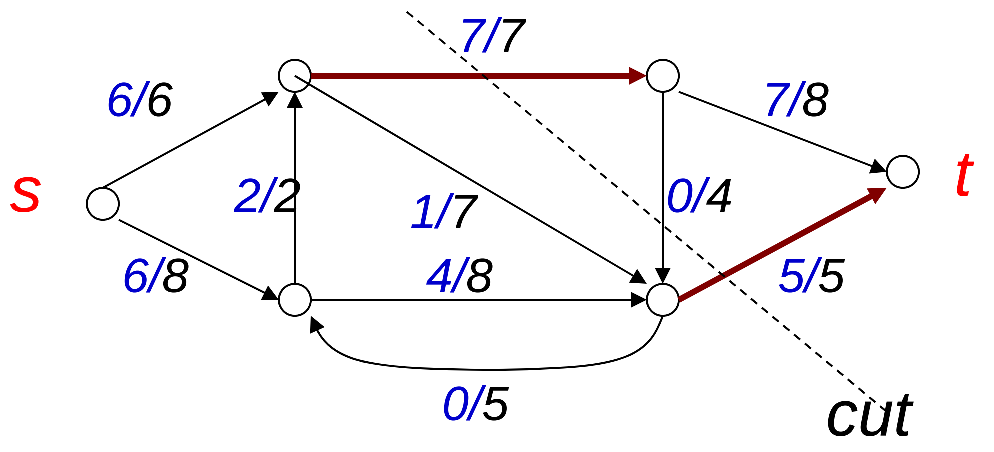
All forward edges of cut saturated, all backward have 0 flow

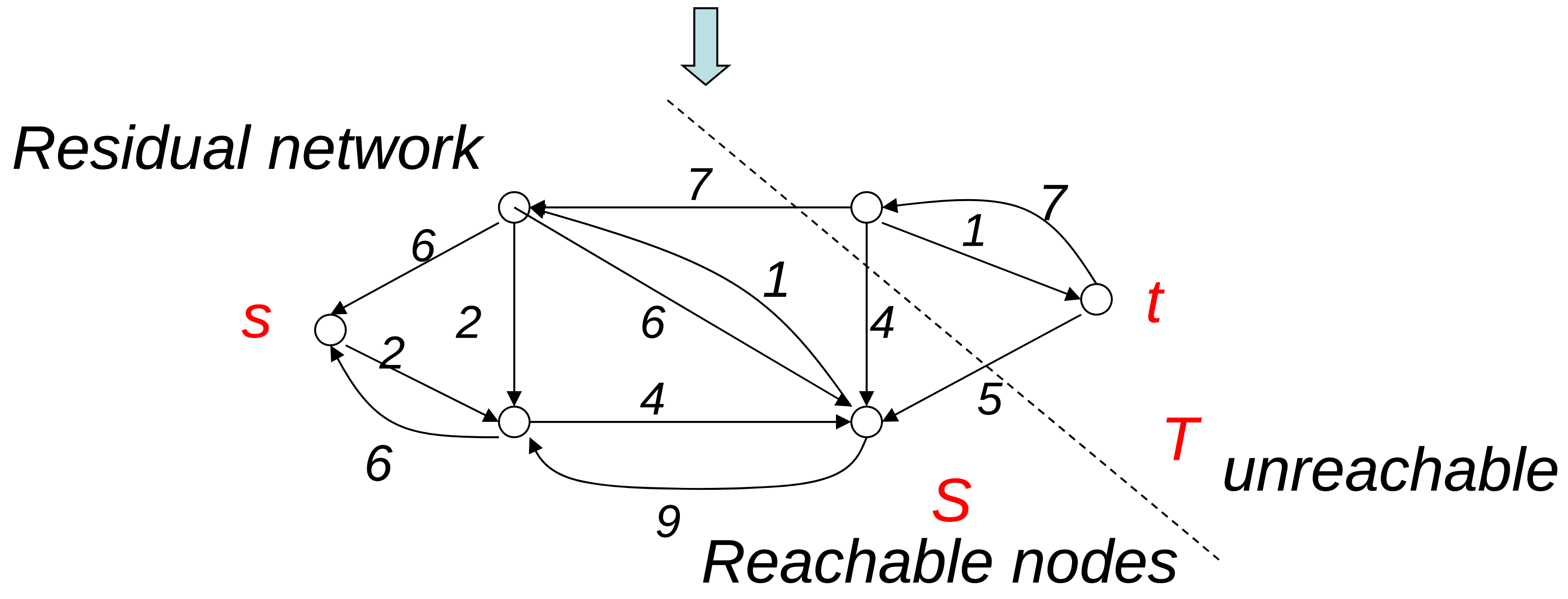
$$|f| = f(S \to T) - f(T \to S) = f(S \to T) = c(S,T)$$

Since every flow has value ≤ c(S,T), flow f is maximum

Max Flow = Min Cut

Example: Max flow = min cut =12





Properties of Max Flow and Ford-Fulkerson Algorithm

INTEGRALITY PROPERTY: If all capacities are integer, then flow is integer at all times

$\Rightarrow$ There is an integral max flow f*

Every iteration increases the value of the flow by at least 1

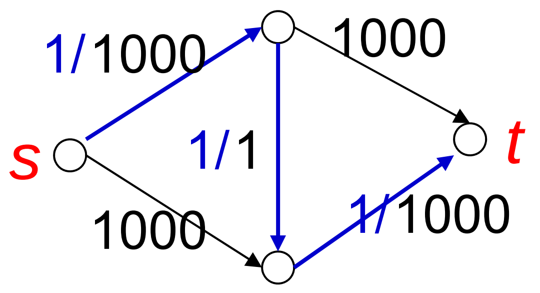
$\Rightarrow$ # iterations $\leq$ |f*|

Upper bound on time O(e |f*|)

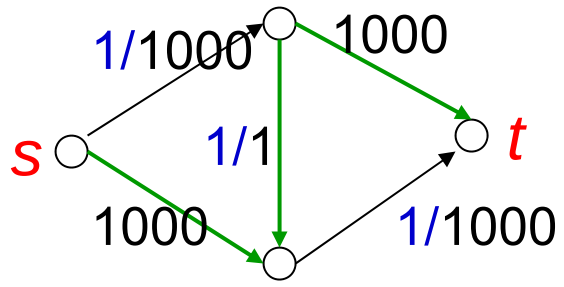
In general, performance depends on the policy for choosing the augmenting path p

Bad Example for FF

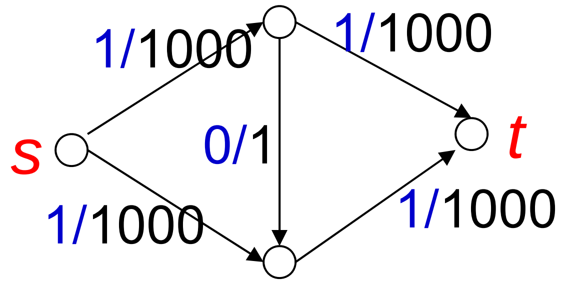
Flow of value 1



augmenting path



New Flow of value 2



Iterate 2000 times to get flow of value 2000

Edmonds-Karp Algorithm

- Use breadth-first search in G_f to compute a *shortest augmenting path* p
- # iterations $\leq$ ne $\Rightarrow$ complexity $O(ne^2)$

- Other faster Algorithms: $O(n^3)$, O(nelogn)

- *Key lemmas of Edmonds-Karp*:

  1. The distances dist(s,v) of all nodes v in the residual network G_f do not decrease with each augmentation

  2. Each edge (u,v) is "critical" edge (has min residual capacity) of an augmenting path < n/2 times.

  Every augmenting path has a critical edge $\Rightarrow$ at most ne/2 augmentations.

Edmonds-Karp Proofs

1. The distances $d_f(v)$= dist(s,v) in G_f of all nodes v from s do not decrease with each augmentation

Proof: flow f , augmenting path p $\rightarrow$ flow f'

Suppose $\exists$v. $d_{f'}(v) < d_f(v)$, and v has minimum $d_{f'}(v)$

Shortest s-v path in $G_{f'}$

$$\begin{tikzcd} s \arrow[r, dashed] & u \arrow[r] & v \end{tikzcd}$$

Choice of v $\Rightarrow d_{f'}(u) \geq d_f(u)$ $\Rightarrow$ no edge (u,v) in G_f $\Rightarrow$ edge (v,u) in p

p shortest path in $G_f \Rightarrow d_f(v) < d_f(u) \leq d_{f'}(u) < d_{f'}(v)$

Edmonds-Karp Proof ctd.

2. Each edge (u,v) is critical (min residual capacity) edge of an augmenting path < n/2 times.

Proof: flow f , augmenting path p with critical edge (u,v)

$d_f(v) = d_f(u)+1$

(u,v) critical in p $\Rightarrow$ (u,v) not present in new residual graph

- Before (u,v) reappears in residual graph, must have

augmentation in some flow f' along path that includes (v,u)

$d_{f'}(u) = d_{f'}(v)+1 \geq d_f(v)+1 = d_f(u)+2$

Hence d(u) increases by 2 between any two times that (u,v) critical $\Rightarrow$ it is critical at most n/2 times.

Extensions, Generalizations

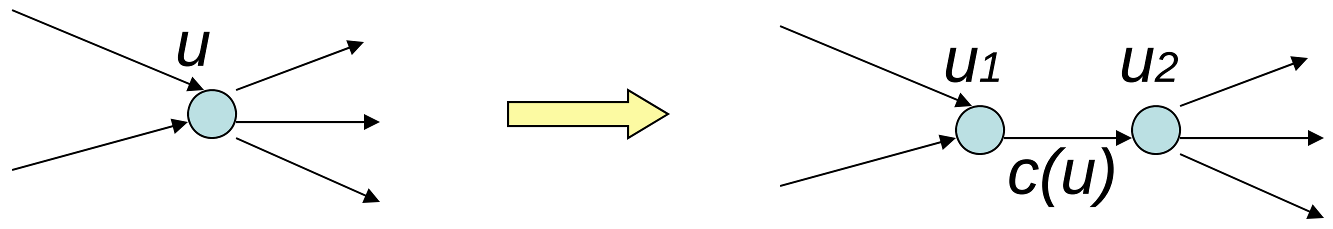
Many variants, extensions can be reduced to the basic max flow and min cut problems

- Undirected Graphs
- Node capacities
- Multiple sources and sinks
- Lower and upper bounds on edge flows

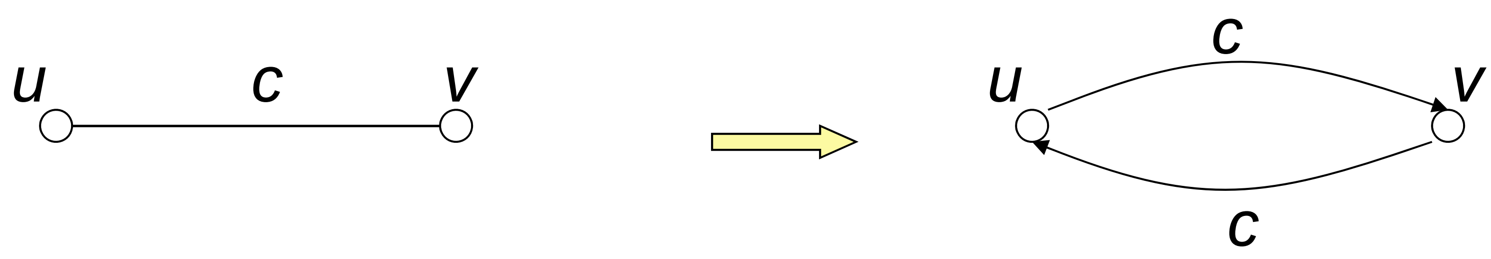
- Other Problems
- Graph connectivity problems
- Maximum Bipartite matching
- Minimum bipartite node cover
- ……

Node capacities

- For each node u, upper bound c(u) on the flow into u (= flow out of u)



Undirected Graphs

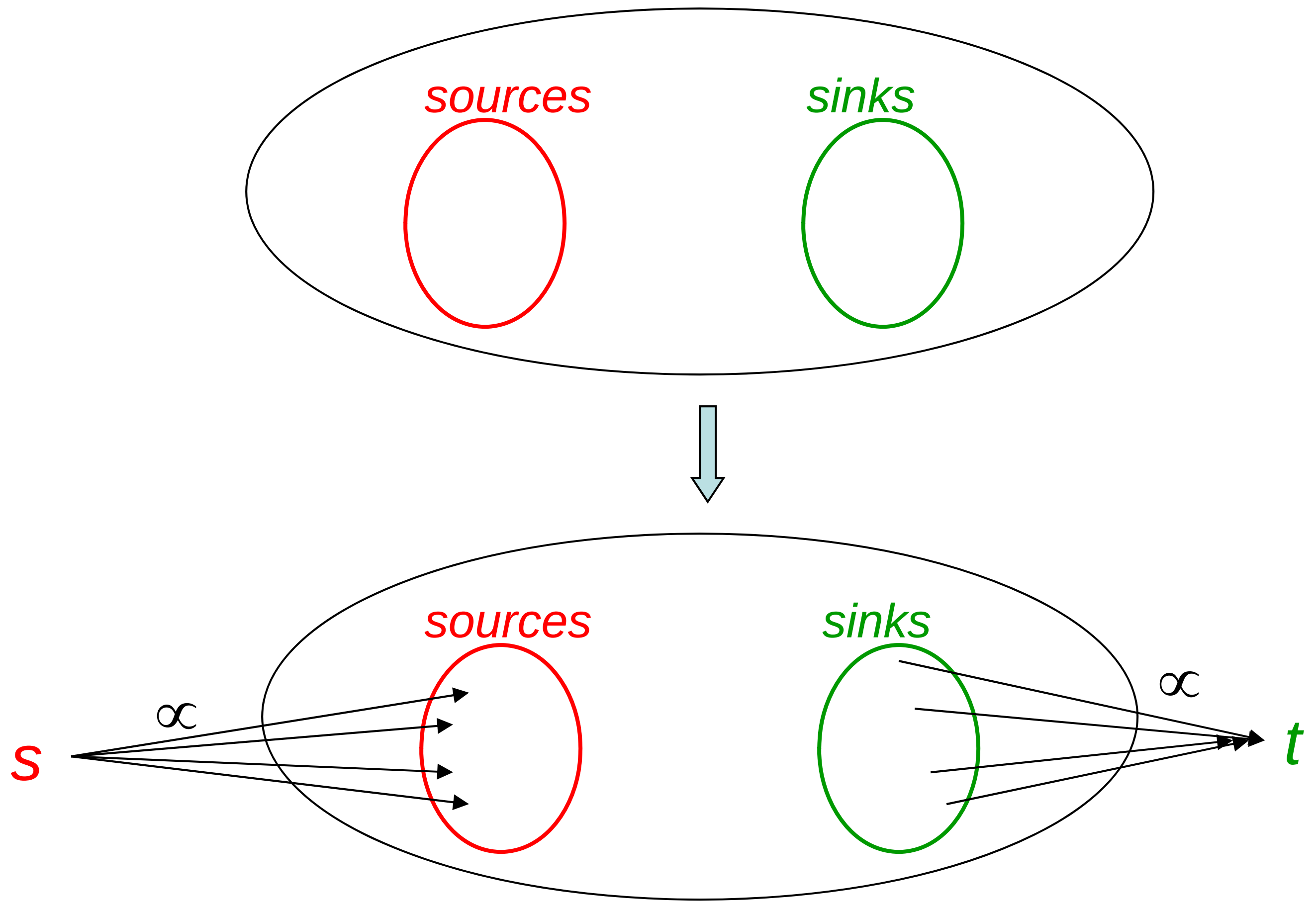


• Without loss of generality the maximum flow in the directed network will not have positive flow on opposite edges (if not, we can cancel the flow on one edge with equal amount of flow on the opposite edge)

⇒ Total traffic on each undirected edge ≤ capacity

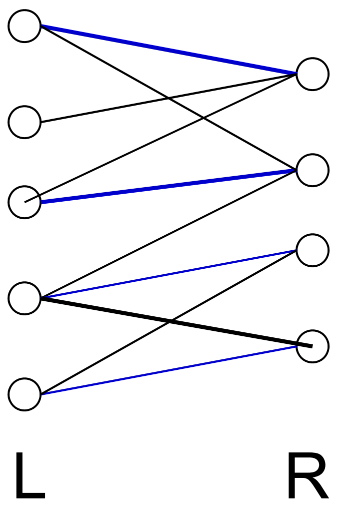
Networks with multiple sources and sinks

Can be reduced to single source - single sink case



If every source (sink) can produce (consume) limited amount of flow, then set correspondingly the capacities of the new edges

Maximum Bipartite Matching



Bipartite undirected graph

$N = L \cup R$

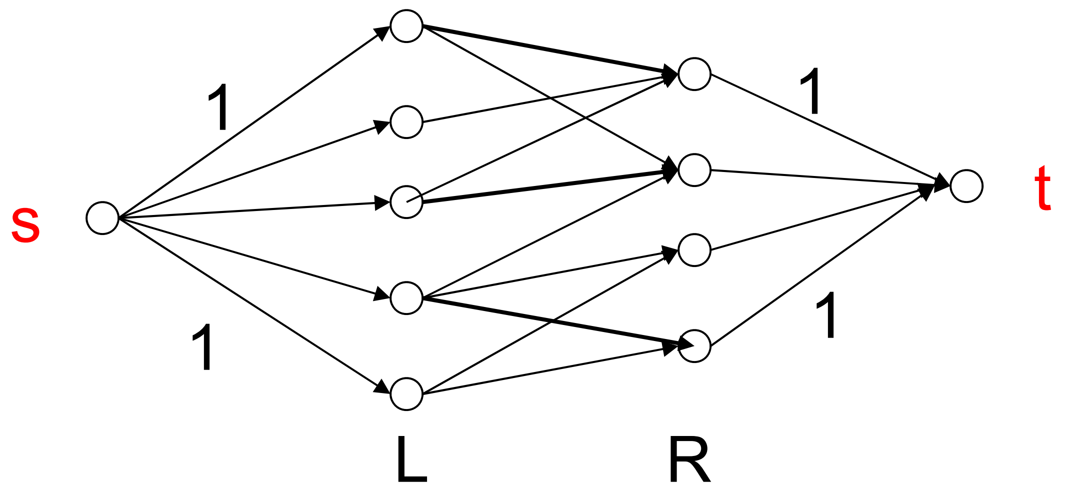
$E \subseteq L \times R$

Matching: Set of disjoint edges (i.e. no common nodes)

(= pairing of some L nodes with distinct R nodes)

Maximum Matching Problem: Find matching with maximum cardinality

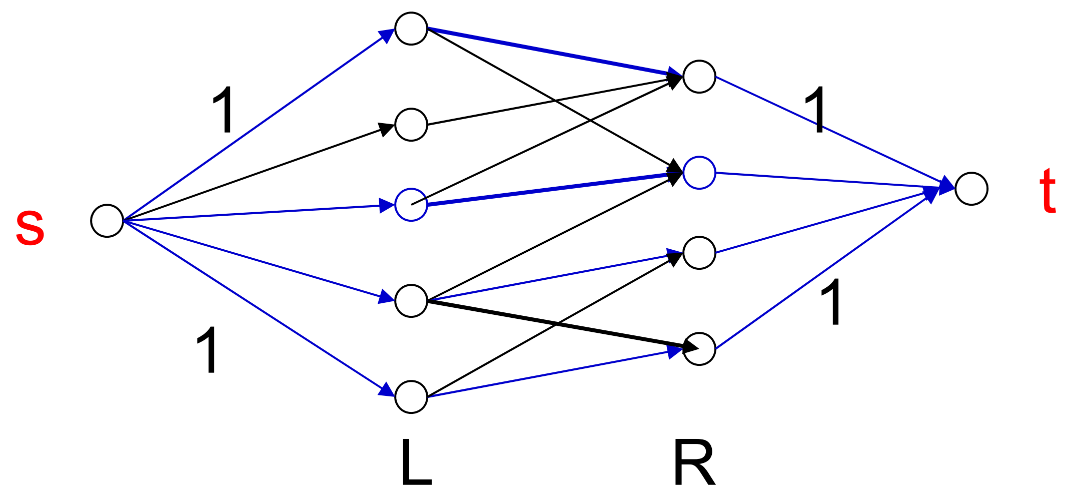
Reduction to max flow problem



All left and right edges have capacity 1

Middle edges can have any capacity ≥1 (eg, 1,2,…, ∝)

Reduction to max flow problem



- Integer capacities ⇒ Integer max flow
- Integer-valued flows = 0 -1 valued flows ↔ matchings
- Max integer flow = max flow ↔ maximum matching

Complexity of Ford-Fulkerson: O(ne)
Hopcroft-Karp: $O(\sqrt{n}\, e)$

- Maximum matching in nonbipartite graphs can be solved in same time, but more complicated (not via flows)

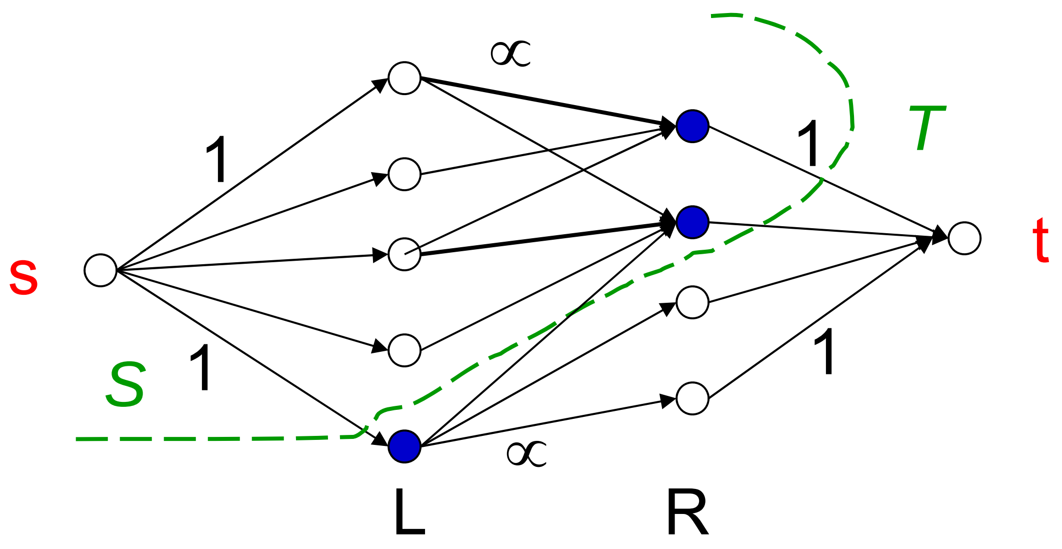
Node Cover problem

- Node cover of undirected graph G is a set C of nodes such that every edge is incident to ("is covered by") some node in C
- Given G, find node cover of minimum cardinality

- |Maximum Matching| ≤ |Minimum Node Cover|

Proof: Every edge of a matching must be covered by a distinct node of node cover

Bipartite Node Cover reduces to s-t Min Cut



- Set capacities of middle edges to ∝

1. Given node cover C, define cut (S,T) where
   $S=\{s\}\cup(L-C)\cup(R\cap C)$, and $T=\{t\}\cup(R-C)\cup(L\cap C)$
2. Given min cut (S,T), define $C=(L\cap T)\cup(R\cap S)$
   Min cut has no ∝ edge $\Rightarrow$ C is a node cover of same size

In bipartite graphs: Maximum matching = Minimum node cover

Linear Programming

Linear Programming

- Variables take values in real numbers
- *General Form*: Maximize or minimize a linear function

  (the *objective function)* subject to a set of linear constraints: linear weak inequalities and equations

$$\max(\min)\sum_{j=1}^{n} c_j x_j$$

$$\text{subject to}\quad \left\{\begin{array}{l} \sum_{j=1}^{n} a_{1j}x_j = b_1 \\ \ldots \\ \sum_{j=1}^{n} a_{ij}x_j \le b_i \\ \ldots \\ \sum_{j=1}^{n} a_{kj}x_j \ge b_k \end{array}\right.$$

$$\ldots$$

Applications

- Manufacturing, Marketing, Finance, Transportation, Telecommunications, …
- Optimal allocation of resources to satisfy constraints and maximize profit or minimize cost
- Can be used also to model many optimization problems in various areas

Max s-t Flow problem as a LP

- LP with edge-flow variables f(u,v), (u,v)$\in$E

- Maximize $\sum_{(s,v)\in E} f(s,v) - \sum_{(v,s)\in E} f(v,s)$

subject to

- f (u,v) ≥0, $\forall$(u,v)$\in$E (nonnegativity)
- f (u,v) ≤ c(u,v) , $\forall$(u,v) $\in$E (Capacity constraints)
- Flow conservation constraints:

$$\sum_{v:(u,v)\in E} f(u,v) - \sum_{v:(v,u)\in E} f(v,u) = 0, \ \forall u \in N - \{s,t\}$$

Example: Minimum Cost Flow problem

- Given flow network G=(N,E,c) with source s, sink t, cost a(u,v)≥0 for each edge (u,v), and flow demand d, find a flow of value d from s to t that minimizes the total cost

- LP with edge-flow variables $f(u,v)$, $(u,v) \in E$

$$
\begin{aligned}
\min \; & \sum_{(u,v)\in E} a(u,v)\cdot f(u,v) \\
\text{s.t.} \; & \sum_{(u,v)\in E} f(u,v) - \sum_{(v,u)\in E} f(v,u) = 0, \; \forall u \in N - \{s,t\} \\
& \sum_{(s,v)\in E} f(s,v) - \sum_{(v,s)\in E} f(v,s) = d \\
& f(u,v) \le c(u,v), \;\; \forall (u,v) \in E \\
& f(u,v) \ge 0, \qquad\;\; \forall (u,v) \in E
\end{aligned}
$$

Can omit (set to 0) the variables f(v,s) if there are edges into s

Integrality property: Integer capacities ⇒ Integer optimal flow

Shortest s-t path = Min cost flow with all caps=1, demand=1

Example: Multicommodity Flow problem

- Flow network G=(N,E,c) with *k* source-sink pairs (s_i, t_i), one for each commodity i=1,…,*k.* (Some nodes may be the source or sink for multiple commodities.)
- The flows of different commodities share the edges; total flow through each edge may not exceed the capacity

Different versions:

1. *Maximization version*: Find flows for the commodities that maximize the sum of all the commodities shipped.
2. *Demands version*: Given demand d_i for each commodity, find a flow that ships d_i units from s_i to t_i

   Also, *Minimum Cost* version: find minimum-cost such flow

Maximum Multicommodity Flow problem

1. Maximum Flow: edge flow variables $f_i(u,v)$, $(u,v) \in E$, $i=1,\ldots,k$

amount of commodity i flowing through edge (u,v)

$$
\begin{aligned}
\max\ & \sum_{i=1}^{k} \Big[\sum_{(s_i,v)\in E} f_i(s_i,v) - \sum_{(v,s_i)\in E} f_i(v,s_i) \Big] \\
\text{s.t.}\ & \sum_{(u,v)\in E} f_i(u,v) - \sum_{(v,u)\in E} f_i(v,u) = 0,\ \ \forall i = 1,\ldots,k,\ \forall u \in N - \{s_i,t_i\} \\
& \sum\nolimits_{i=1}^{k} f_i(u,v) \le c(u,v),\quad \forall (u,v) \in E \\
& f_i(u,v) \ge 0,\quad \forall i = 1,\ldots,k,\quad \forall (u,v) \in E
\end{aligned}
$$

• No integrality property:

Even if capacities are integer, optimal flow may not be integer

Multicommodity Flow problem with demands

2. Demands & costs: variables $f_i(u,v)$, $(u,v)\in E$, $i=1,\ldots,k$

$$
\begin{aligned}
\min\ & \sum_{i=1}^{k} \sum_{(u,v)\in E} a(u,v)\cdot f_i(u,v) \\
\text{s.t.}\ & \sum_{(u,v)\in E} f_i(u,v) - \sum_{(v,u)\in E} f_i(v,u) = 0,\ \ \forall i=1,\ldots,k,\ \forall u\in N-\{s_i,t_i\} \\
& \sum_{i=1}^{k} f_i(u,v) \le c(u,v),\ \ \forall (u,v)\in E \\
& \sum_{(s_i,v)\in E} f_i(s_i,v) - \sum_{(v,s_i)\in E} f_i(v,s_i) = d_i,\ \ \ \forall i=1,\ldots,k \\
& f_i(u,v) \ge 0,\ \ \forall i=1,\ldots,k,\ \ \forall (u,v)\in E
\end{aligned}
$$

• Even if capacities and demands are integer and no costs, it may be that the only way to satisfy the demands is with a fractional flow

Example: Fractional Knapsack Problem

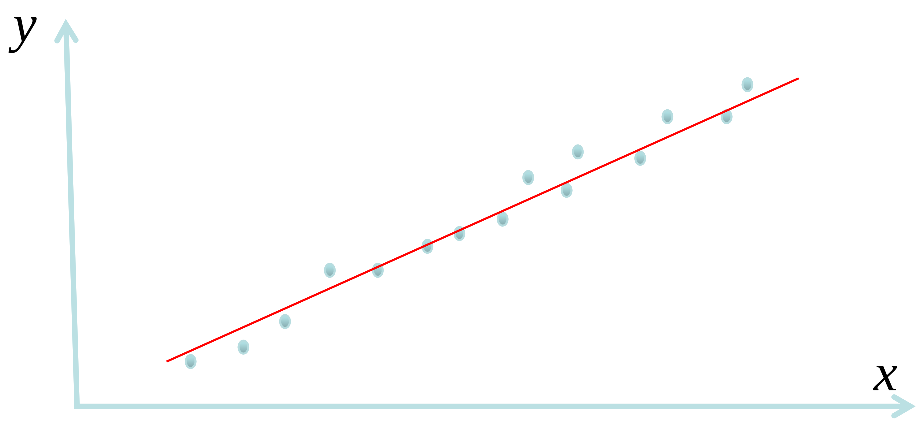
- Given n items with values $v_1,\dots,v_n$ and weights $w_1,\dots,w_n$, and a knapsack of weight capacity B, choose quantities $x_1,\dots,x_n$ of the items (possibly fractional) to put in the knapsack so that they all fit and have maximum total value
- LP formulation: variables $x_1,\dots,x_n$

$$\max \sum_{i=1}^{n} v_i x_i$$

$$\text{s.t} \quad \sum_{i=1}^{n} w_i x_i \le B$$

$$0 \le x_i \le 1, \quad i = 1,\dots,n$$

Example: Fitting a line to points



Given points (x_i, y_i) on plane find a line $y = ax + b$ of best fit

- Minimize least square error

  Closed form solution from calculus

- Minimize sum of absolute errors

  Can formulate as LP

LP for line fitting

$$\min \sum_{i=1}^{n} e_i$$

subject to

$$e_i \geq ax_i + b - y_i$$

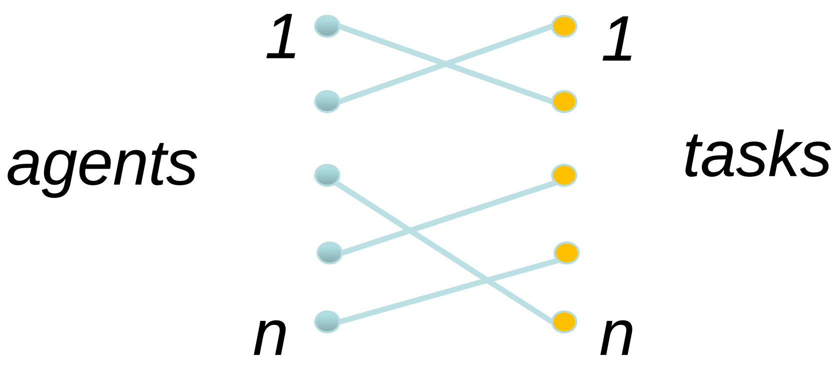
$$e_i \geq -(ax_i + b - y_i)$$

Variables: $a, b, \{e_i \mid i = 1, \ldots, n\}$

In optimal solution: $e_i = |ax_i + b - y_i|$

Example: Assignment Problem

- *n* agents and *n* tasks, value v_{ij} if we assign agent *i* to task *j*.
- We want to find a 1-1 assignment of agents to tasks that maximizes the total value



LP for Assignment Problem

- Variables x_{ij} indicating whether agent i is assigned task j

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} v_{ij} x_{ij}$$

subject to:

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \text{for all } \; i = 1, \ldots, n$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad \text{for all } \; j = 1, \ldots, n$$

$$x_{ij} \geq 0, \quad \text{for all } \; i, j = 1, \ldots, n$$

This LP has always an integral optimal solution, i.e. an optimal solution with x_{ij} *= 0 or 1 for all* i, j

Three Possibilities for LP

- Infeasible: Constraint set has no feasible solution
- Unbounded: No finite optimum: objective function can be made arbitrarily “good” (large for a maximization problem, small for minimization)
- Finite optimum: There is an optimal solution

  (note: the feasible solution set itself may be unbounded in some directions)

LP modeling - example

- A steel company can produce two products: bands, coils

Profit: \$25/ton for bands, \$30/ton for coils

Maximum demand/week: 6,000 for bands, 4,000 for coils

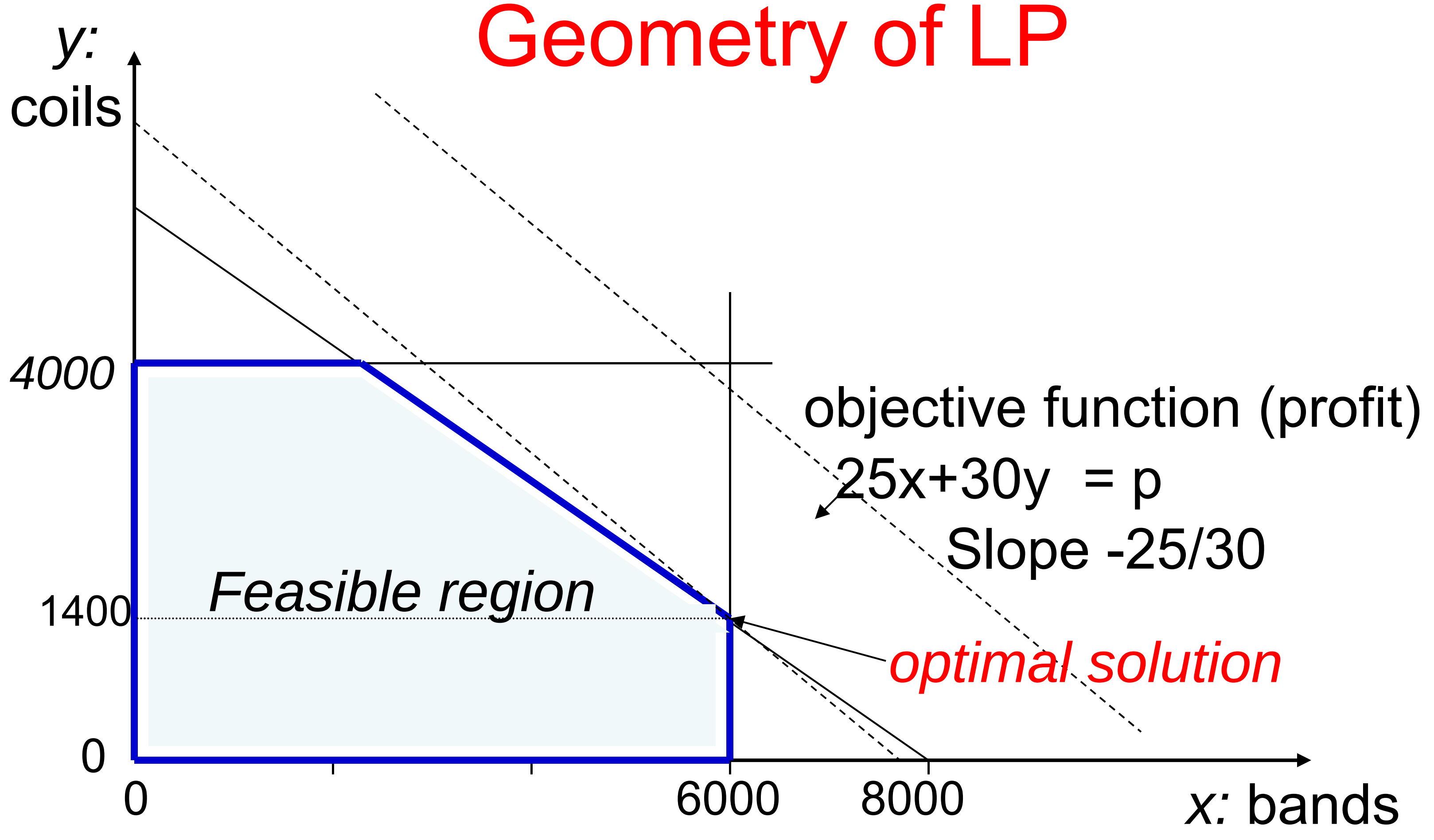
Production Rate: for bands 200 tons/hour, coils: 140 tons/hr
Week = 40 hours of production

How many tons of each to maximize profit?

Variables: x = #tons of bands per week, y = #tons of coil/ week

$$\begin{aligned} &\max 25x + 30y \\ &\text{s.t.}\quad \frac{1}{200}x + \frac{1}{140}y \le 40 \\ &\qquad 0 \le x \le 6000 \\ &\qquad 0 \le y \le 4000 \end{aligned}$$

Geometry of LP



Feasible region: a polyhedron

Optimal solution: a vertex

Vertices of Polyhedron

- Vertex is determined by the intersection of n (=dimension) linearly independent hyperplanes (tight constraints)
- If m constraints and n variables $\rightarrow$ at most $\binom{m}{n}$ vertices
- If all input coefficients in the constraints and the objective function are rationals p/q, where p,q are integers with w bits, then the coordinates of the vertices are also rationals p'/q' where p',q' have polynomial (in n,w) # of bits

Algorithms for Linear Programming

- Simplex (Dantzig, 1947)

  Method: Starts at a vertex and keeps moving to better adjacent vertex until it reaches an optimum

- In practice, works very well
- In worst case, many “pivoting rules” (rules for choosing which better adjacent vertex to move to) can lead to exponential (in n,m) number of iterations
- Open if there is a pivoting rule that guarantees polynomial time (polynomial number of iterations)

Algorithms for Linear Programming ctd.

- Ellipsoid Algorithm (Khachian, 1979)

  Worst case polynomial time (in n,m,w), but not practical

- Karmakar's Algorithm & Interior Point methods (1984)

  Worst case polynomial time (in n,m,w),

  competitive in practice