Lecture: Analysis of Algorithms (CS583 - 002)[1]

Amarda Shehu

Fall 2017

[1] Some material adapted from Kevin Wayne's Algorithm Class @ Princeton

1. Maximum Flow and Minimum Cut Problem
   - Flow Networks
   - Minimum Cut
   - Of Cuts and Flows
   - Maximum Flow
   - Weak Duality
   - Strong Duality
   - Maximum Flow Algorithm: Ford-Fulkerson
   - Improving Ford-Fulkerson: Capacity Scaling
2. Graph Applications
   - Bipartite Matching: Max Flow Application
   - Clustering: MST Application
   - Motion Planning: Shortest Path Application

Max Flow and Min Cut

Exhibition:

- Very rich algorithmic problems
- Cornerstones in combinatorial optimization
- Exhibit mathematical duality

Applications

- Data mining
- Project selection
- Airline scheduling
- Bipartite matching
- Baseball elimination
- Image segmentation
- Network connectivity
- Threading hydrophobic/hydrophilic residues in a protein 3D conformation
- Network reliability
- Distributed computing
- Egalitarian stable matching
- Security of statistical data
- Network intrusion detection
- Multi-camera scene reconstruction

Some History: Soviet Rail Network, 1955

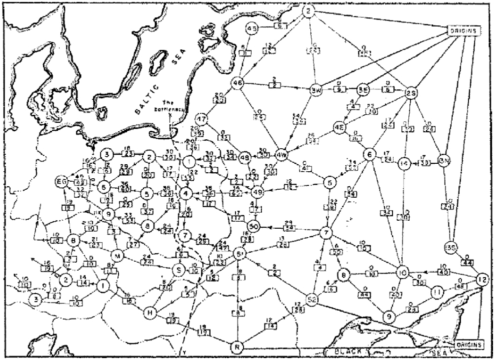
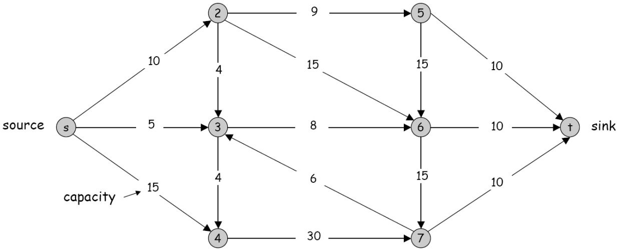


Figure: On the history of transportation and maximum flow problems. Alexander Schrijver in Math Programming, 91:3, 2002

Flow Networks

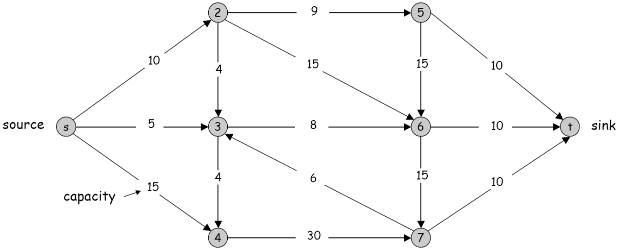
- Network flow is an advanced branch of graph theory
- A weighted directed graph with two special vertices
- The **source** vertex, which has no incoming edges
- The **sink** vertex, which has no outgoing edges
- These are respectively labeled s and t



Flow Networks

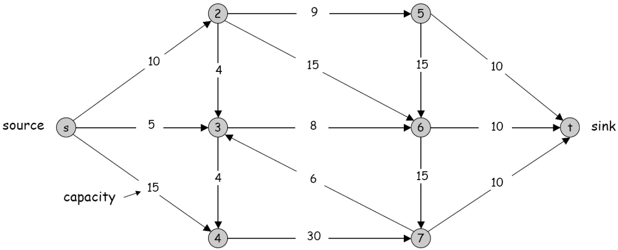
Flow network:

- $G = (V, E)$ is a directed graph with no parallel edges
- Nodes are junctions and edges are pipes
- A pipe allows water/material to flow only one way
- $c(e)$ is the capacity associated with an edge e



Finding Maximum Flow in a Flow Network

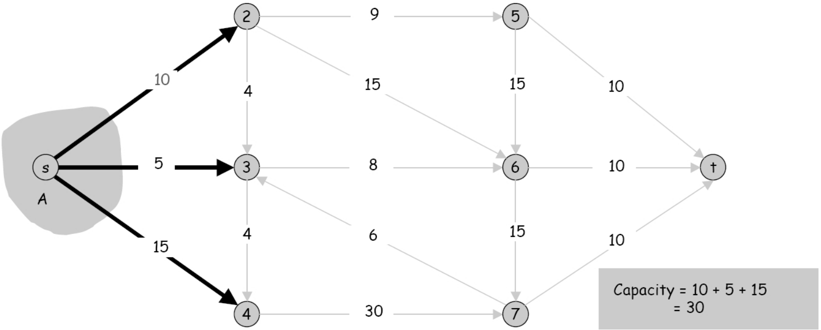
- Pour an infinite amount of water/material in source
- Goal: find maximum flow, the maximum amount of material/water that will reach the sink
- Max flow and min cut are dual concepts
- Setup for Ford-Fulkerson method to find max flow



Talking About Cuts

Definition: An $s - t$ cut is a partition (A, B) of V with $s \in A$ and $t \in B$

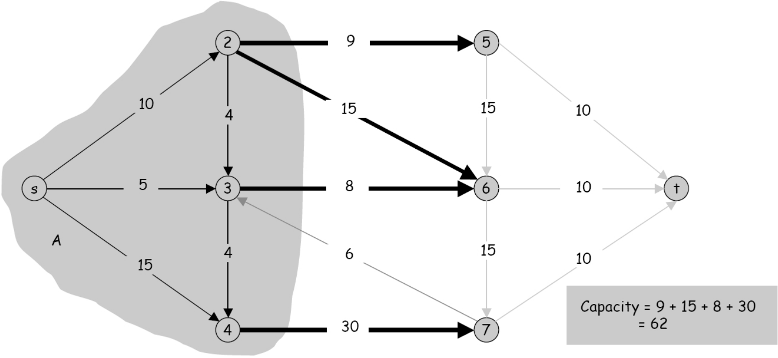
Definition: The capacity $\mathrm{cap(A,B)} = \sum_{\{\mathrm{e=(u,v):u \in A, v \in B}\}} \mathrm{c(e)}$



Another Valid $s-t$ Cut

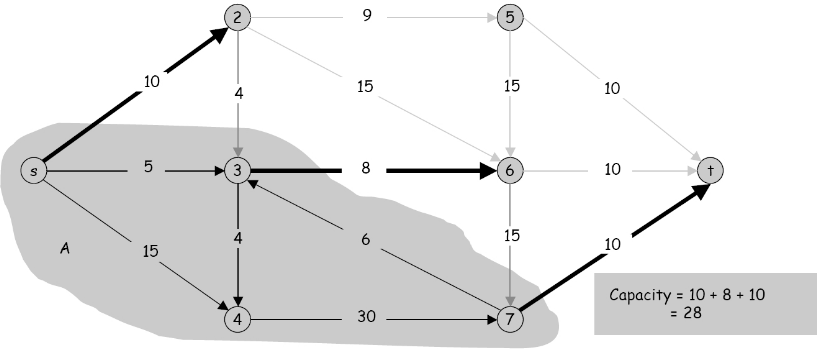
Definition: An $s-t$ cut is a partition (A, B) of V with $s \in A$ and $t \in B$

Definition: The capacity $\mathrm{cap(A,B)} = \sum_{\mathrm{\{e=(u,v):u\in A, v\in B\}}} \mathrm{c(e)}$



The Minimum Cut Problem

Min $s - t$ Cut Problem: Find an $s - t$ cut of minimum capacity

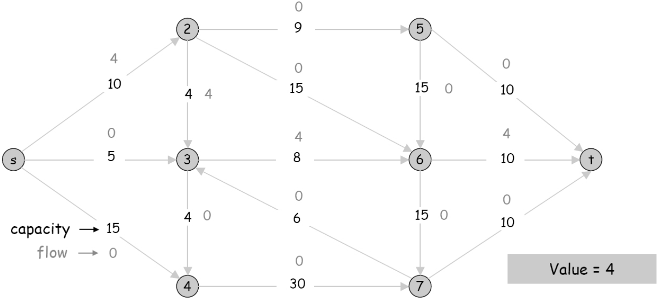


From an $s - t$ Cut to an $s - t$ Flow

Definition: An $s - t$ flow is a function $f : E \rightarrow \mathcal{R}$ that satisfies:

- $\forall e \in E$: $0 \leq f(e) \leq c(e)$ [flow cannot exceed capacity]
- $\forall v \in V - \{s, t\}$: $\sum_{e=(*,v)} f(e) = \sum_{e=(v,*)} f(e)$ [conservation]

Definition: The value of a flow f is: $\nu(f) = \sum_{e=(s,*)} f(e)$

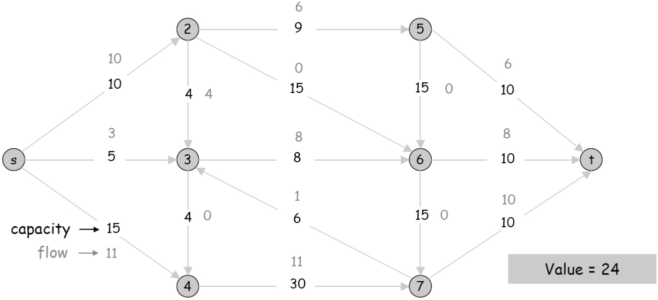


From an $s - t$ Cut to an $s - t$ Flow

Definition: An $s - t$ flow is a function $f : E \rightarrow \mathcal{R}$ that satisfies:

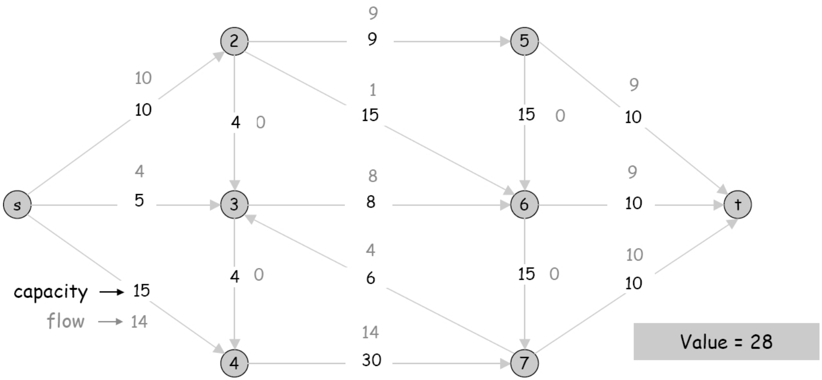
- $\forall e \in E$: $0 \leq f(e) \leq c(e)$ [flow cannot exceed capacity]
- $\forall v \in V - \{s, t\}$: $\sum_{e=(*,v)} f(e) = \sum_{e=(v,*)} f(e)$ [conservation]

Definition: The value of a flow f is: $\nu(f) = \sum_{e=(s,*)} f(e)$



The Maximum Flow Problem

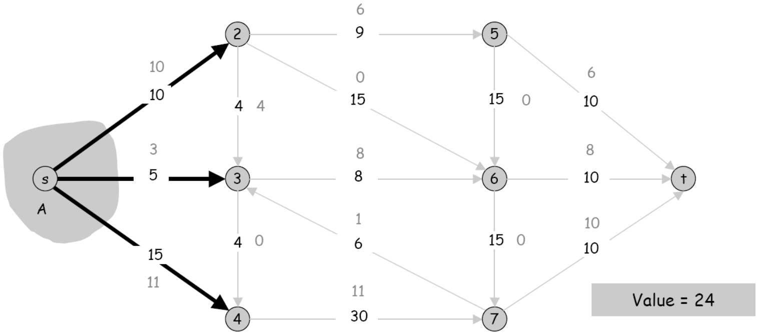
Max Flow Problem: Find $s - t$ flow f of maximum flow value $\nu(f)$



Net Flow Across a Cut

Let (A, B) be any $s - t$ cut. The net flow sent across the cut is:

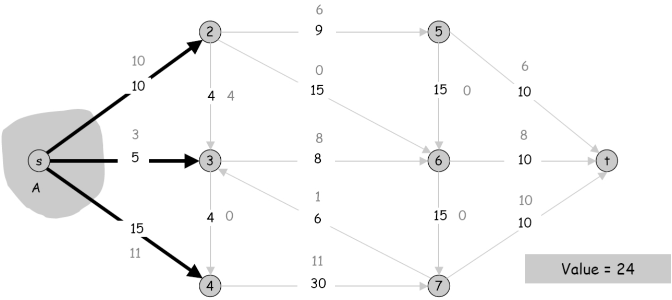
$$\sum_{e \text{ out of A}} f(e) - \sum_{e \text{ in to A}} f(e)$$



Flow and Cut Duality

Flow value lemma: Let f be any $s-t$ flow, and let (A, B) be any $s-t$ cut. Then, the net flow sent across the cut is equal the amount $\nu(f)$ leaving s:

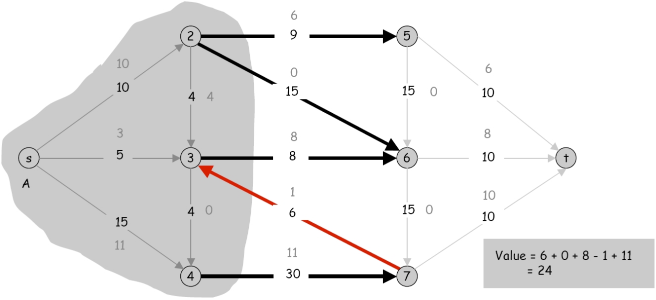
$$\sum_{e \text{ out of A}} f(e) - \sum_{e \text{ in to A}} f(e) = \nu(f)$$



Flow and Cut Duality

Flow value lemma: Let f be any $s-t$ flow, and let (A,B) be any $s-t$ cut. Then, the net flow sent across the cut is equal the amount $\nu(f)$ leaving s:

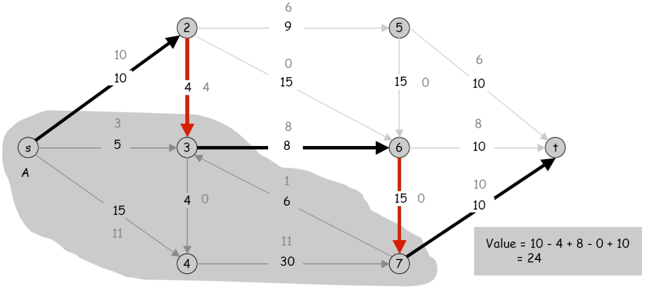
$$\sum_{\text{e out of A}} f(e) - \sum_{\text{e in to A}} f(e) = \nu(f)$$



Flow and Cut Duality

Flow value lemma: Let f be any $s - t$ flow, and let (A, B) be any $s - t$ cut. Then, the net flow sent across the cut is equal the amount $\nu(f)$ leaving s:

$$\sum_{\text{e out of A}} f(e) - \sum_{\text{e in to A}} f(e) = \nu(f)$$



Proof of Flow Cut Duality

Flow value lemma: Let f be any $s-t$ flow, and let (A, B) be any $s-t$ cut. Then, the net flow sent across the cut is equal the amount $\nu(f)$ leaving s:

$$\sum_{\text{e out of A}} f(e) - \sum_{\text{e in to A}} f(e) = \nu(f)$$

Proof: Due to flow conservation, $\sum_{\{e=(v,*)\}} f(e) = \sum_{\{e=(*,v)\}} f(e)$ for all vertices $v \in V - \{s, t\}$. So:

$$\sum_{v \in V - \{s,t\}} [\sum_{\{e=(v,*)\}} f(e) - \sum_{\{e=(*,v)\}} f(e)] = 0$$

By definition, $\nu(f) = \sum_{e=(s,*)} f(e)$. Adding 0 to both sides yields:

$$\begin{aligned} \nu(f) &= \textstyle\sum_{e=(s,*)} f(e) + 0 \\ &= \textstyle\sum_{e=(s,*)} f(e) + \sum_{v \in V - \{s,t\}} [\sum_{\{e=(v,*)\}} f(e) - \sum_{\{e=(*,v)\}} f(e)] \end{aligned}$$

Proof of Flow Cut Duality Continued

So, at this point we are summing up the net flow of vertices $v \in V - \{t\}$.

$$\sum_{e=(s,*)} f(e) + \sum_{v \in V - \{s,t\}} [\sum_{\{e=(v,*)\}} f(e) - \sum_{\{e=(*,v)\}} f(e)]$$

Let's define an arbitrary cut (A, B). The vertices $v \in V - \{t\}$ will be split into those with both in and out edges either complete inside A or completely inside B, and those with edges connecting A to B.

Due to flow of conservation, summing up the net flow over vertices with both in and out edges completely in A or completely in B will give 0.

So, in the above equation we are left with summing up only the net flow over vertices that have edges connecting A to B:

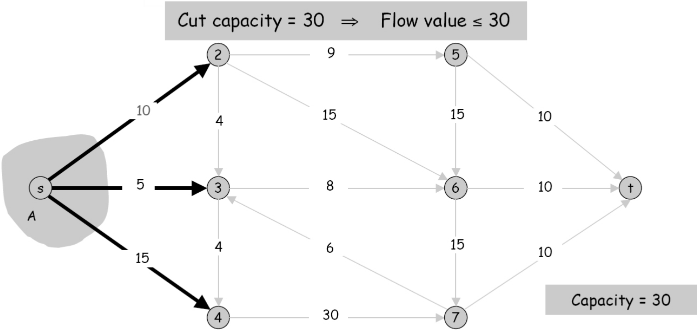
$$\begin{aligned} \nu(f) &= \sum_{e=(s,*)} f(e) + \sum_{v \in V - \{s,t\}} [\sum_{\{e=(v,*)\}} f(e) - \sum_{\{e=(*,v)\}} f(e)] \\ &= \sum_{e \text{ out of A}} f(e) - \sum_{e \text{ in to A}} f(e) \end{aligned}$$

Reflecting on the Implications of the Flow Cut Duality

- The previous proof says: Given *any valid* flow and *any valid* cut, the flow value is equal to the net flow sent across the cut
- So, over all possible flows f and all possible cuts (A, B) $\nu(f) = \sum_{e \text{ out of A}} f(e) - \sum_{e \text{ in to A}} f(e)$
- What is the maximum flow value that can be achieved?
- There is a cut (among all possible valid cuts that can be defined) that limits the maximum flow

Flows and Cuts: Weak Duality

Weak Duality: Let f be any $s-t$ flow, and let (A, B) be any $s-t$ cut. Then the value $\nu(f)$ of the flow is at most the capacity $\mathrm{cap(A, B)}$ of the cut:

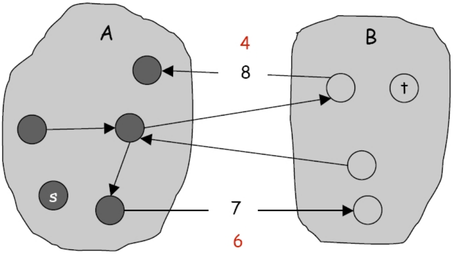


Flows and Cuts: Weak Duality

Weak Duality: Let f be any s-t flow. For any s-t cut (A, B), $\nu(f) \leq \mathrm{cap(A,B)}$

Proof:

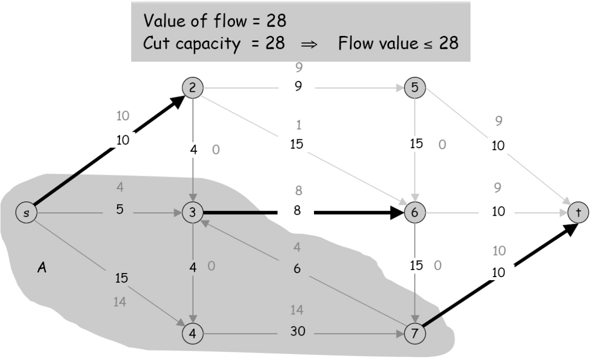
$$\begin{aligned} \nu(f) &= \sum_{e \text{ out of A}} f(e) - \sum_{e \text{ in to A}} f(e) \\ &\leq \sum_{e \text{ out of A}} f(e) \\ &\leq \sum_{e \text{ out of A}} c(e) \\ &= \mathrm{cap(A,B)} \end{aligned}$$



Implications: Max flow is the dual of the min cut problem.

Certificate of Optimality

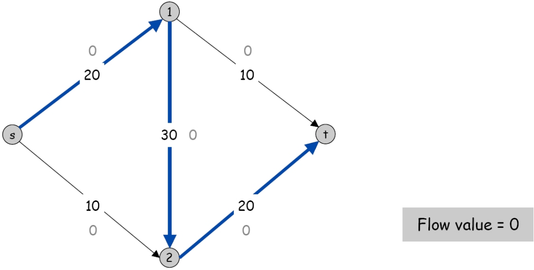
Corollary: Let f be any $s-t$ flow, and let (A, B) be any cut. If $\nu(f) = \mathrm{cap(A, B)}$, then f is a max flow and (A, B) is a min cut.



Designing a Max Flow Algorithm

Greedy Algorithm

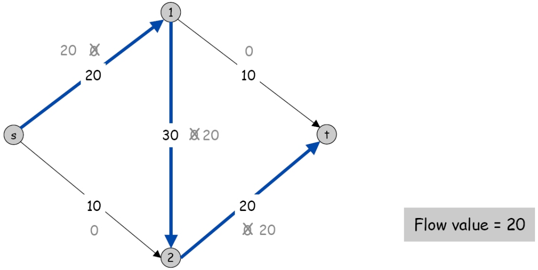
- Start with $f(e) = 0$ for every edge $e \in E$
- Find an $s - t$ path P where each edge has $f(e) < c(e)$
- Augment flow along path P
- Repeat until stuck



Designing a Max Flow Algorithm

Greedy Algorithm

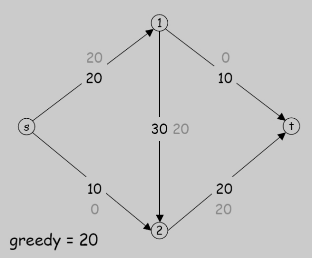
- Start with $f(e) = 0$ for every edge $e \in E$
- Find an $s - t$ path P where each edge has $f(e) < c(e)$
- Augment flow along path P
- Repeat until stuck

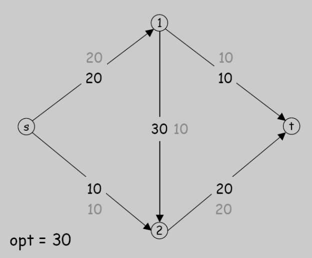


Designing a Max Flow Algorithm

Greedy Algorithm

- Start with $f(e) = 0$ for every edge $e \in E$
- Find an $s - t$ path P where each edge has $f(e) < c(e)$
- Augment flow along path P
- Repeat until stuck (locally optimal is not globally optimal)



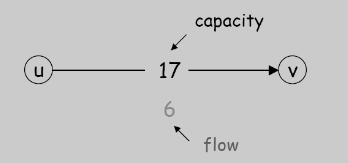


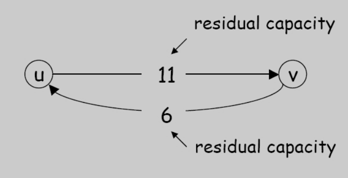
Order of Paths is Important

- We cannot guarantee which path we will find first
- If we pick wrong path first, whole algorithm goes wrong
- Key now is idea of pushing back flow
- If we have x units of water flowing in the pipe (u, v), then we can pretend there is a pipe (v, u) with capacity x when we are trying to find a path from s to t
- This is maintained through a residual graph

Residual Graph

- A residual graph G_f allows to keep track of which paths remain from s to t along which one can push more flow.
- The idea of "can push more flow" is kept through residual capacities in G_f.
- G_f in the beginning is a copy of the given input graph $G = (V, E)$
- When a flow $f(e)$ is pushed along $e = (u, v)$, G_f contains two edges:
  - (u, v) with residual capacity $c(e) - f(e)$
  - (v, u) with residual capacity $f(e)$





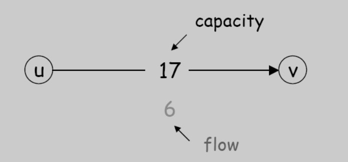
Residual Graph

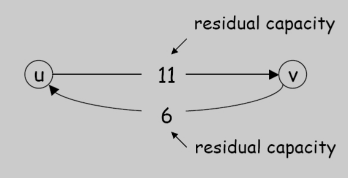
Residual graph: $G_f = (V, E_f)$

- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$
- Associate residual capacity $c_f(e) > 0$
- $\forall e = (u, v) \in E(G)$ with $c(e), f(e)$:

$$\begin{cases} e, c_f(e) = c(e) - f(e) & \text{if } f(e) < c(e) \\ e^R, c_f(e) = f(e) & \text{else} \end{cases}$$

- G_f tracks edges of G can admit more flow
- A path $P = s \rightsquigarrow t$ in G_f is an augmenting path in G with respect to f
- $\nu(f)$ can be increased by $c_f(P) = \min_{e \in P} c_f(e)$ [$c_f(P)$ is bottleneck of P]





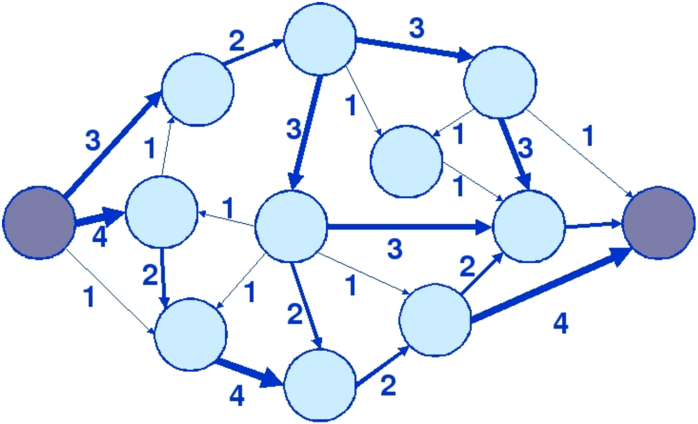
Ford-Fulkerson: An Augmenting Path Algorithm

```
Augment(f, c, P) {
   b ← bottleneck(P)
   foreach e ∈ P {
      if (e ∈ E) f(e) ← f(e) + b        forward edge
      else       f(eR)← f(eR) - b       reverse edge
   }
   return f
}
```

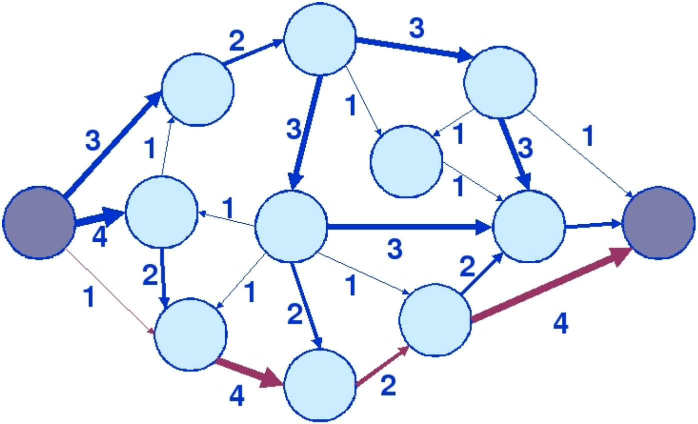
```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Gf ← residual graph

   while (there exists augmenting path P) {
      f ← Augment(f, c, P)
      update Gf
   }
   return f
}
```

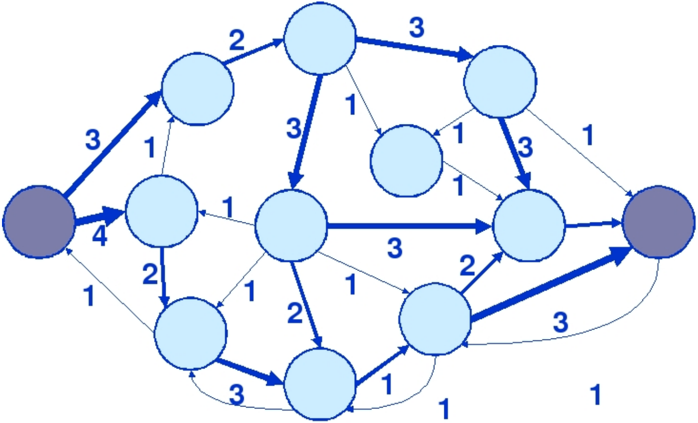
Ford-Fulkerson: Trace



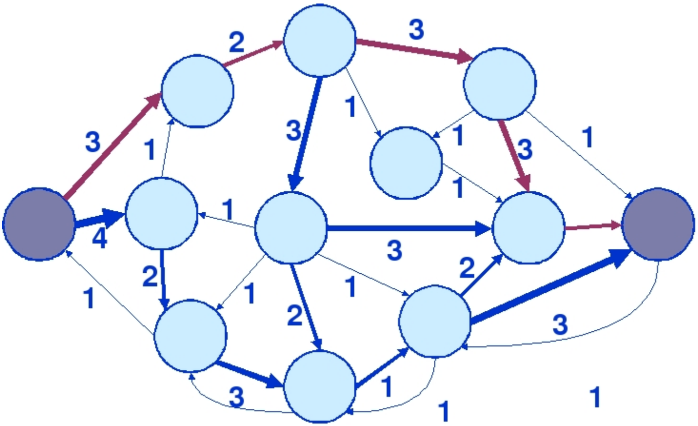
Ford-Fulkerson: Trace



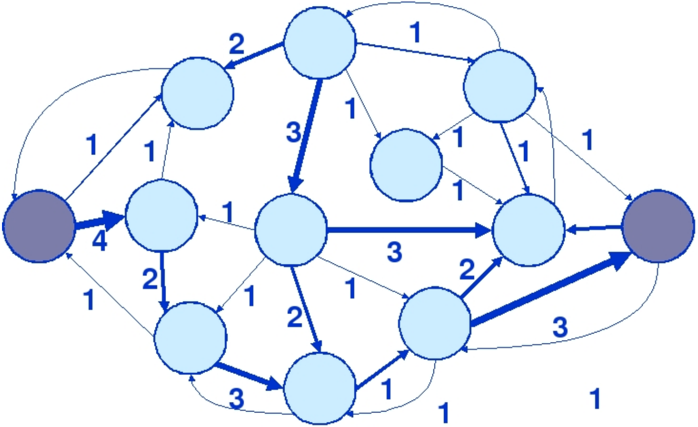
Ford-Fulkerson: Trace



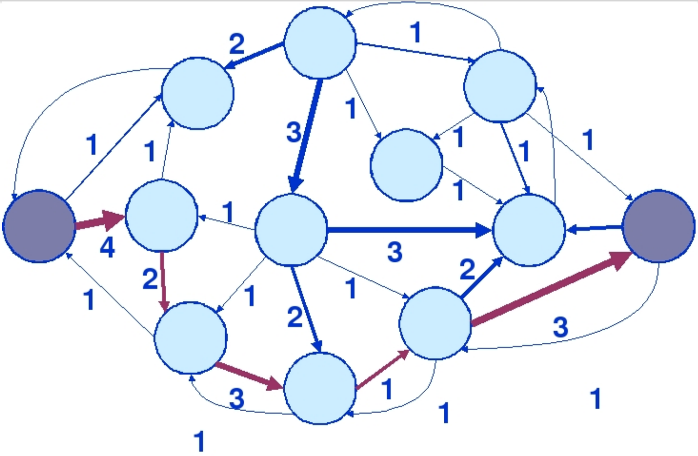
Ford-Fulkerson: Trace



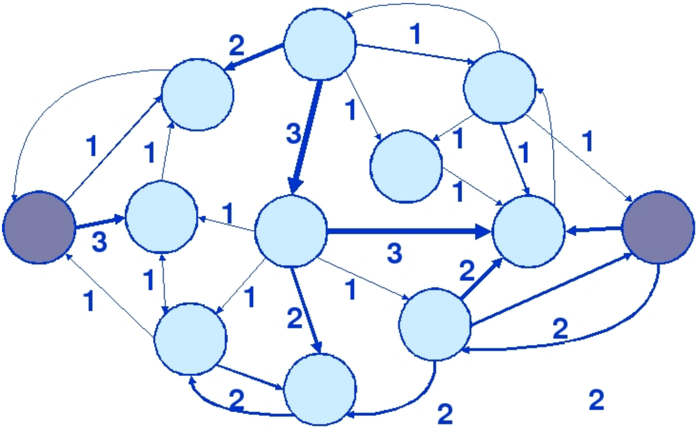
Ford-Fulkerson: Trace



Ford-Fulkerson: Trace



Ford-Fulkerson: Trace



Max-Flow Min-Cut Theorem

Augmenting Path Theorem: f is a max flow iff there are no augmenting paths

Max-flow Min-cut Theorem: The value of the max flow is equal to the value of the min cut [Elias-Feinstein-Shannon 1956, Ford-Fulkerson 1956]

Proof: Both simultaneously by showing:

- (i) There exists a cut (A, B) such that $\nu(f) = \mathrm{cap(A, B)}$
- (ii) Flow f is a max flow
- (iii) There is no augmenting path relative to f

(i) $\Rightarrow$ (ii): From weak duality lemma

(ii) $\Rightarrow$ (iii): Let f be a flow. If there is an augmenting path, then f can be improved by sending flow along path.

Max-Flow Min-Cut Theorem (Continued)

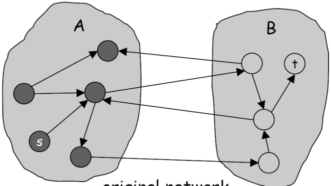
(iii) $\Rightarrow$ (i):

- Let f be a flow with no augmenting paths
- Let A be set of vertices reachable from s in residual graph
- $s \in A$ by definition of A
- $t \notin A$, otherwise t would be reachable from s in G_f

Since there are no augmenting paths in G_f, the residual capacities $c_f(e) = 0$ for all edges out of A.

That is, $\forall e$ out of A, $f(e) = c(e)$.

Since the flow of each edge out of A is the capacity of that edge, then $\nu(f) = \mathrm{cap}(\mathrm{A}, \mathrm{B})$



original network

Ford-Fulkerson: Correctness and Analysis

Assumption: All capacities are integers between 1 and C

Invariant: Every $f(e)$ and $c_f(e)$ remains an integer throughout the execution

Theorem: The algorithm runs in $O(|E| \cdot f^*)$, where f^* is the maximum flow

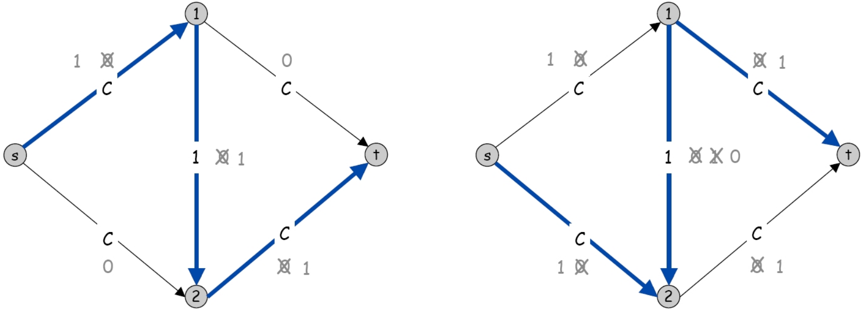
Proof: Since each augmentation increases value by at least 1, the algorithm iterates over at most f^* augmentations. At each augmentation, the flow is pushed over at most $|E|$ edges ($|E_f| \leq 2 \cdot |E|$).

Integrality Theorem: If all capacities are integers, then there exists a max flow for which every flow value $f(e)$ is an integer

Proof: Follows from invariant, given that the algorithm terminates

Ford-Fulkerson: Correctness and Analysis

If maximum capacity is C, the algorithm takes C iterations in the worst case.



Choosing Good Augmenting Paths

Choose good augmenting paths

- Some choices lead to exponential algorithms
- Clever choices lead to polynomial algorithms
- If capacities are irrational, algorithm not guaranteed to terminate

Choose augmenting paths that:

- Can be found efficiently
- Result in few iterations

Such paths have: [Edmunds-Karp 1972, Dinitz 1970]

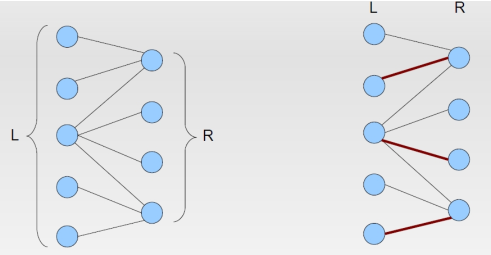
- Max bottleneck capacity
- Sufficiently large bottleneck capacity
- Fewest number of edges

Edmunds-Karp

- Ford-Fulkerson is more of a template than an algorithm
- When capacities are integers, Ford-Fulkerson guaranteed to terminate in $O(|E| \cdot f)$ time, where f is max flow value
- With irrational flow values, algorithm may never terminate
- Edmunds-Karp: a variation of the Ford-Fulkerson's algorithm with guaranteed termination and a $\mathrm{O}(|V| \cdot |E|^2)$ runtime independent of the maximum flow value

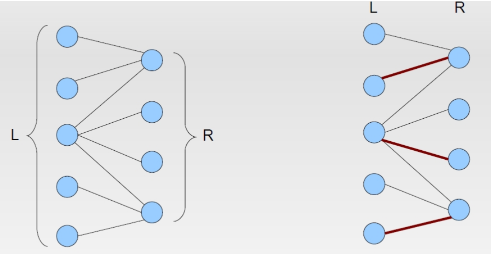
Bipartite Matching

- Matching a set of machines L with a set of tasks R that need to be performed simultaneously
- An edge (u, v) denotes machine u can execute task v
- Goal is to maximize number of tasks



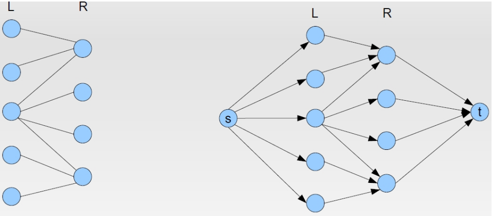
Bipartite Matching

- Given a bipartite graph $G = (L \cup R, E)$, find a maximal matching, a subset of the edges, no two of which share an endpoint
  - Dating agency, matching women L with men R
  - An edge (u, v) indicates u is compatible with v
  - Goal is to maximize number of matches



Bipartite Matching Reduces to Maximum Flow

- Add a source s, edges (s, l) for $l \in L$, capacity 1
- Add a sink t, edges (r, t) for $r \in R$, capacity 1
- Direct edges in G from L to R, capacity 1
- Integral flows correspond to matchings
- Ford-Fulkerson takes time $O(|V| \cdot |E|)$ since $f \leq |V|$



Clustering: Kruskal's Application

Clustering: Given a set U of n objects labeled $p_1, p_2, \ldots, p_n$, classify them into coherent groups (objects are photos, documents, micro-organisms, gene expression data, events, etc.)

Distance function: Measures "closeness" of two objects

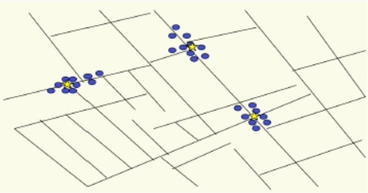


Figure: Outbreak of cholera deaths in London in 1850s (HP Labs)

Clustering of Maximum Spacing

k-clustering: divide objects into k non-empty groups

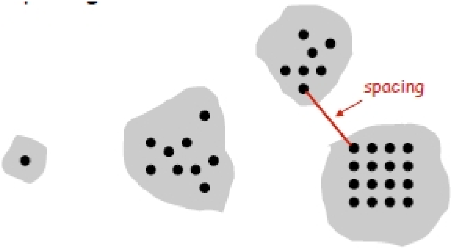
Distance function: satisfies some properties (metric)

- $d(p_i, p_j) = 0$ iff $p_i = p_j$ (identity)
- $d(p_i, p_j \geq 0)$ (non-negative)
- $d(p_i, p_j) = d(p_j, p_i)$ (symmetry)

K-Clustering of Maximum Spacing

Spacing: min distance between any pair of points in different clusters

Clustering of maximum spacing: Given an integer k, find a k-clustering of maximum spacing



Kruskal-like Algorithm

Single-link k-clustering algorithm

- Form a graph $G = (U, \emptyset)$ corresponding to $|U| = n$ clusters
- Find closest pair of objects s.t. each object is in a different cluster, and add an edge between them
- Repeat $n - k$ times until there are exactly k clusters

Key observation: Kruskal-like, except that it stops when there are k connected components

Remark: equivalent to finding an MST and deleting its $k - 1$ most expensive edges

Motion Planning: Dijkstra's Application

- Goal: plan motions of a robot in a cluttered workspace
- Build roadmap/graph of free configuration space of the robot
- Query roadmap for shortest, smoothest paths that allow a robot to get from a start to a goal configuration

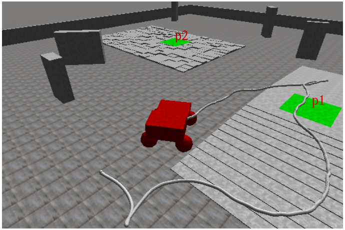


Figure: Erion Plaku at Catholic University is developing algorithms that plan paths for car-like robots in cluttered environments. ©E. Plaku.