Algorithmic Problem Solving

Exercise 05 & Graph II

Herman Appelgren

based on slides by Fredrik Heintz and Fredrik Präntare

Dept of Computer and Information Science

Linköping University

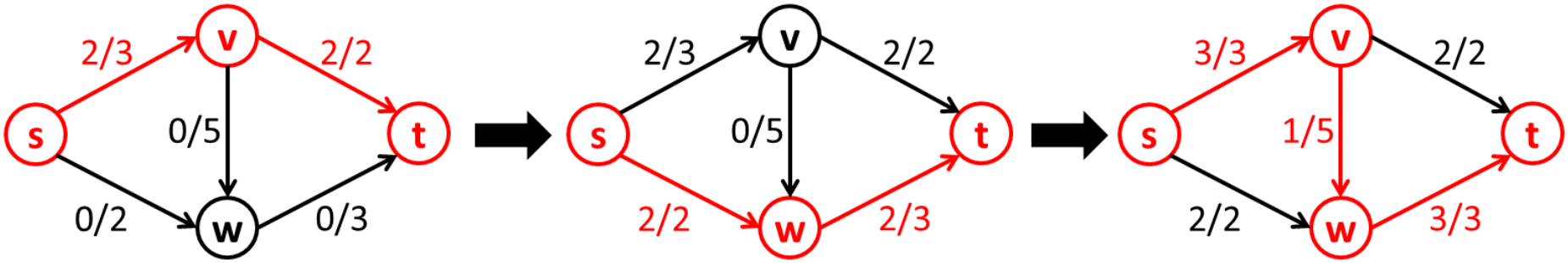
Outline

- ~~**This Week's Problems**~~
  ~~*(Getting Gold, Frogger, Killing Aliens in a Borg Maze, Proving Equivalences)*~~
- Network flow
  - Max Flow (lab 2.6)
  - Min Cut (lab 2.7)
  - Min Cost Max Flow (lab 2.8)

Network Flow

- ***Motivating problem 1*:** Two computers are communicating with each other using a network of routers and cables. Each connection has a set bandwidth. How should we route traffic in the network to maximize data throughput?
- ***Motivating problem 2:*** What connections in the network above are limiting the flow? Alternatively, if we get to increase the bandwidth of some connections, which should be chosen to increase the data throughput?

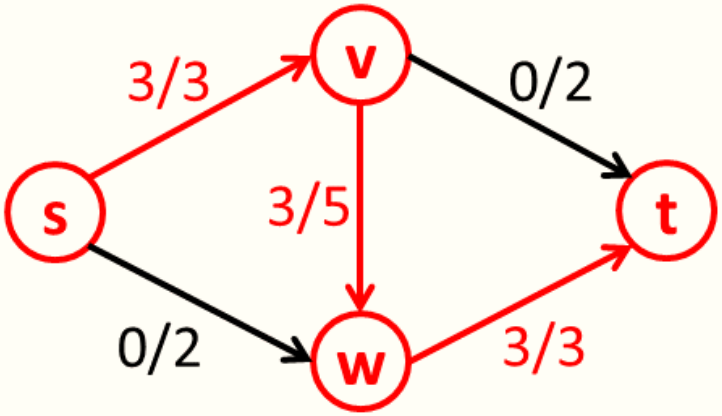
Network Flow



P_1 = [s, v, t]

P_2 = [s, w, t]

P_3 = [s, v, w, t]



Suboptimal solution with ***blocking flow.***

Network Flow

- A ***network*** is a directed graph $G = (V, E)$ with a ***source*** vertex $s \in V$ and a ***sink*** vertex $t \in V$. Also, each edge $(u, v) \in E$ has a ***capacity*** denoted by $c(u, v)$.

  If $(u, v) \notin E$, it is often useful to define $c(u, v) = 0$.

- In a network flow problem, we assign a ***flow*** $f(u, v)$ to all edges $(u, v) \in E$ that satisfy the following properties:
  - **Capacity constraint**: $f(u, v) \leq c(u, v)$ and $f(u, v) \geq 0$ for all $u, v \in V$.
  - **Conservation**: $\sum_{v \in V} f(u, v) = 0$ for all $u \in V \setminus \{s, t\}$.
- The ***flow value*** $F(s)$ from source s is defined as:

$$F(s) = \sum_{v \in V} f(s, v)$$

Ford Fulkerson's Method

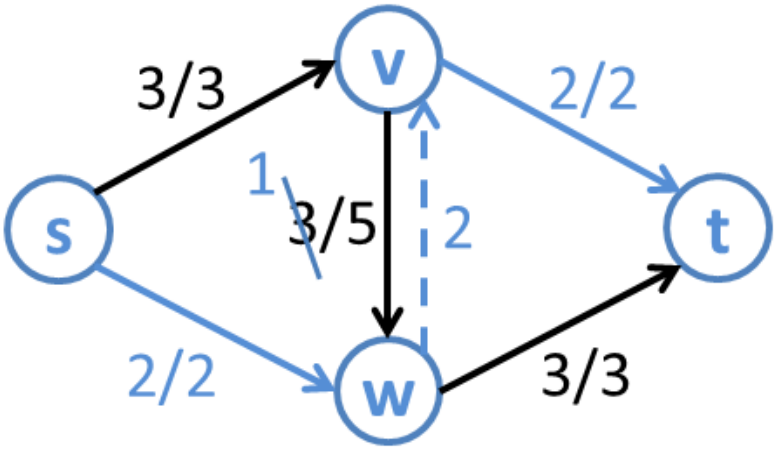
- One Solution: Ford Fulkerson's Method

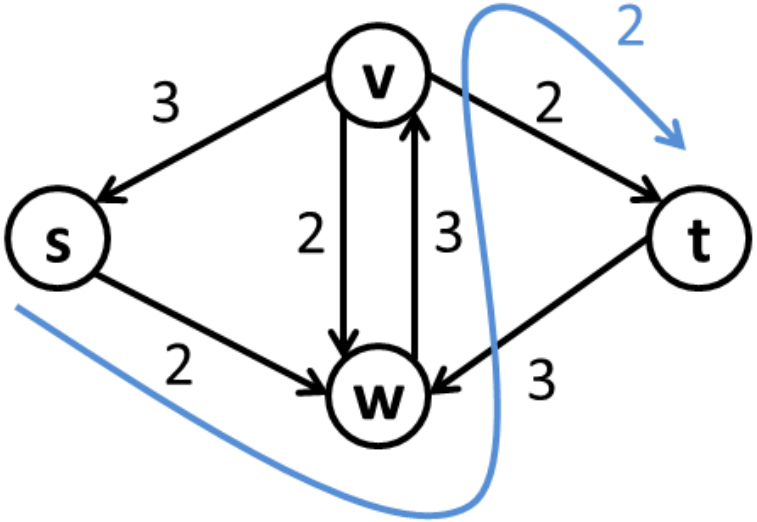

– A surprisingly **simple** *iterative* algorithm

Send a flow *f* through path *p* whenever there exists an **augmenting path** *p* from *s* to *t*

Ford Fulkerson's Method

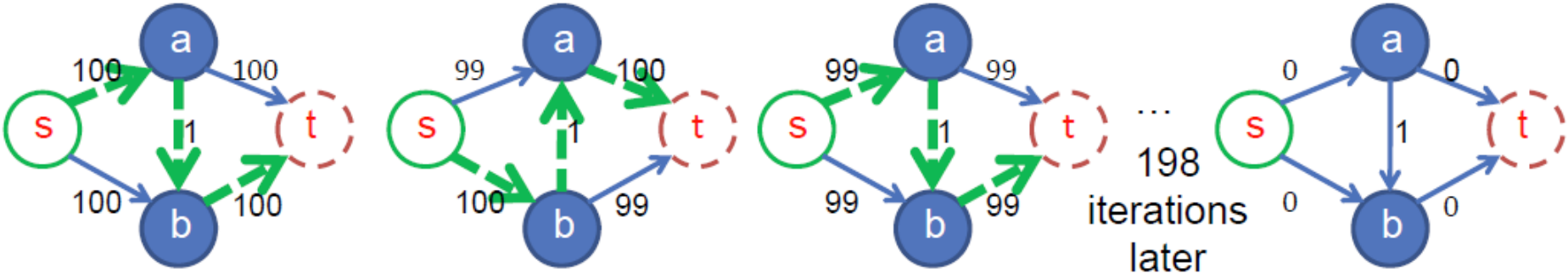
- The ***residual capacity*** of an arc is the difference between an arc's capacity and its flow: $r(u, v) = c(u, v) - f(u, v)$
- To model this residual capacity, we introduce ***back edges*** that can reverse "bad choices of flow".
- A ***residual network*** is a network with back edges (and residual capacities).
- An ***augmenting path*** is a path from s to t in a residual network that we can add positive flow to.





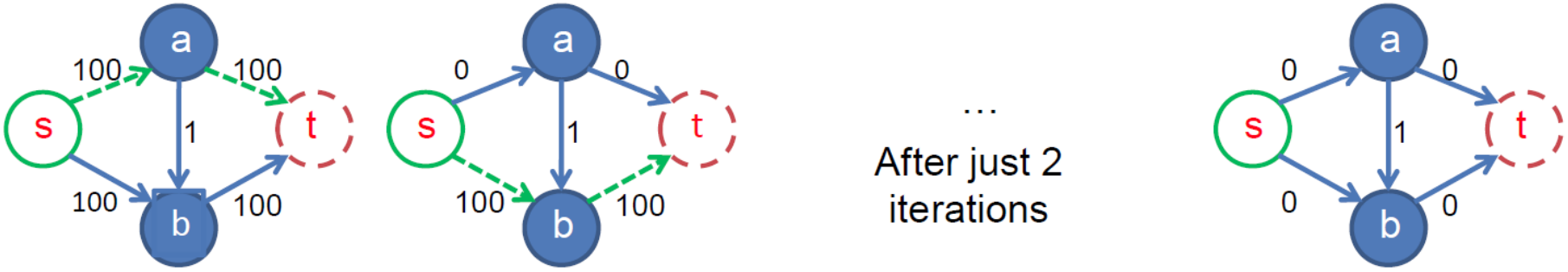
Ford Fulkerson's Method

- ***Ford Fulkerson's method***: Add flow through augmenting paths until no more augmenting paths exists.
- DFS implementation of Ford Fulkerson's runs in $O(f_{max}E)$.
- Very slow on certain types of graphs.



The Edmond-Karp Algorithm

- However, BFS implementation runs in $O(VE^2)$.
- This BFS implementation of Ford-Fulkerson's method is called ***Edmond-Karp's algorithm***.



Maximum Flow Algorithms

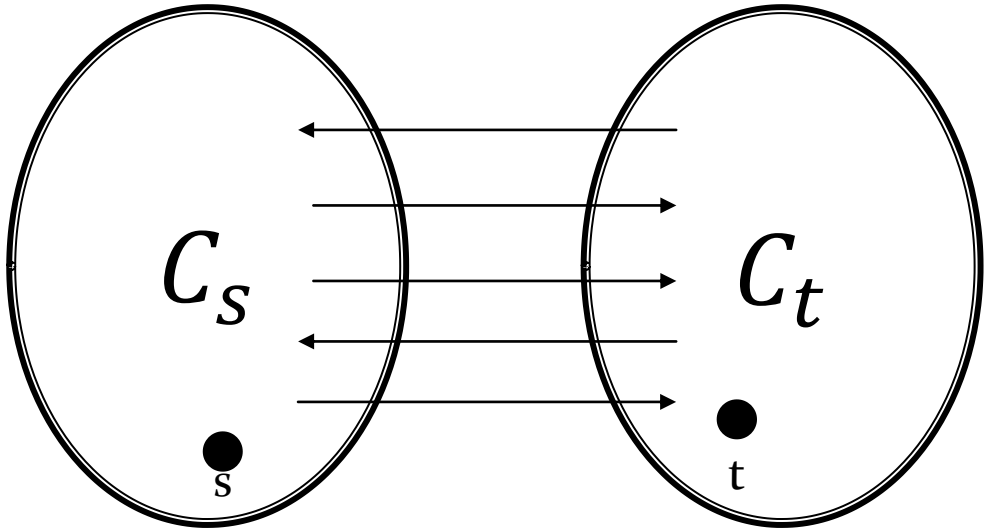
- Ford-Fulkerson with DFS $O(f_{max}E)$
- ***Edmond-Karp*** *(i.e. Ford-Fulkerson with BFS)* $O(VE^2)$
- Dinic's $O(V^2E)$
- Push-relabel (preflow-push) $O(V^3)$
- Binary blocking flow algorithm
  $O(min(V^{2/3}, E^{1/2})\ E\ log(V^2/E)\ log(f_{max}))$

Dinic's Algorithm

- The idea is to reduce our max flow problem to the simple case where all edge capacities are either 0 or 1 (Gabow in 1985 and Dinic in 1973):
  - Scale the problem down by rounding off lower order bits.
  - Solve the rounded problem.
  - Scale the problem back up, add back the bits we rounded off, and fix any errors in our solution.
- In the specific case of the maximum flow problem, the algorithm is:
  - Start with all capacities in the graph at 0.
  - Shift in the higher-order bit of each capacity. Each capacity is then either 0 or 1.
  - Solve this maximum flow problem.
  - Repeat this process until we have processed all remaining bits.
- To scale back up:
  - Start with the maximum flow for the scaled-down problem. Shift the bit of each capacity by 1, doubling all the capacities. If we then double all our flow values, we still have a maximum flow.
  - Restores the lower order bits that we truncated. Find augmenting paths in the residual network to re-maximize the flow.

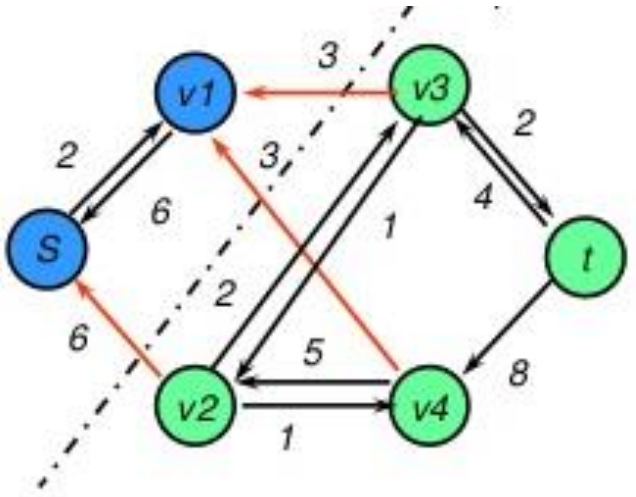
Minimum Cut

- An $s - t$ cut of a network is a partition of its vertices V into 2 groups: C_s and $C_t = V \setminus C_s$, such that $s \in C_s$ and $t \in C_t$.
  - The ***flow*** along cut $C = (C_s, C_t)$ is defined as:
    $$f(C) = \sum v \in C_s \sum w \in C_t \, f(v, w)$$
  - The ***capacity*** of a cut is defined as:
    $$c(C) = \sum v \in C_s \sum w \in C_t \, c(v, w)$$
- A ***minimum cut*** is a cut with minimum possible capacity.



Minimum Cut

- To find a minimum $s - t$ cut $C = (C_s, C_t)$ of G, compute the maximum flow and find the set of vertices reachable from s in the residual graph, this is the set C_s.
- C_s represents the vertices that appear before the closest "bottleneck" (or "choke") that prevents us from adding more positive flow from s to t.
- The ***max-flow min-cut theorem*** states that the maximum flow is equal to the minimum capacity over all $s - t$ cuts.



Minimum Cost Maximum Flow

- Extend the definition of a network flow with a cost per unit of flow on each edge: $k(v, w) \in R$, where $(v, w) \in E$.
- The cost of a flow f is defined as: $k(f) = \sum e \in E\ f(e) \cdot k(e)$
- A minimum cost maximum flow of a network $G = (V, E)$ is a maximum flow with the smallest possible cost.
  - Note that edges in the residual graph of a network need to have their costs determined carefully. Consider an edge (v, w) with capacity $c(v, w)$, cost per unit flow $k(v, w)$. Let $f(v, w)$ be the flow of the edge. Then the residual graph has two edges corresponding to (v, w). The first edge is (v, w) with capacity $c(v, w) - f(v, w)$ and cost $k(v, w)$, and second edge is (w, v) with capacity $f(v, w)$ and cost $-k(v, w)$.
  - It's clear that minimum cost maximum flow generalizes maximum flow by assigning a cost to every edge.
  - It also generalizes shortest path: if we set each cost equal to its corresponding edge length while assigning the same capacity to every edge.
- The maximum flow with minimum cost can be found using a variation of Edmond-Karp's, in which we use Dijkstra instead of BFS.

Network Construction

- Many more complex networks can be reduced to the single-source, single-sink networks we've considered so far.
- Multiple sources/sinks
  - Create a super-source/sink with infinite capacity edges to the sources/sinks.
- Vertex capacities
  - Split each vertex into two vertices and add a bi-directional edge with the vertex capacity between them. Remember to change the edges to the vertex.
- Multigraphs
- Bipartite Matching

Summary

- Residual Graphs, Augmenting Paths, Flow Networks
- Max Flow (lab 2.6)
- Min Cut (lab 2.7)
- Max Flow Min Cut Theorem
- Min Cost Max Flow (lab 2.8)
- Network Construction