Advanced Algorithmic Problem Solving

Le 5 – Graphs part II

Fredrik Heintz
Dept of Computer and Information Science
Linköping University

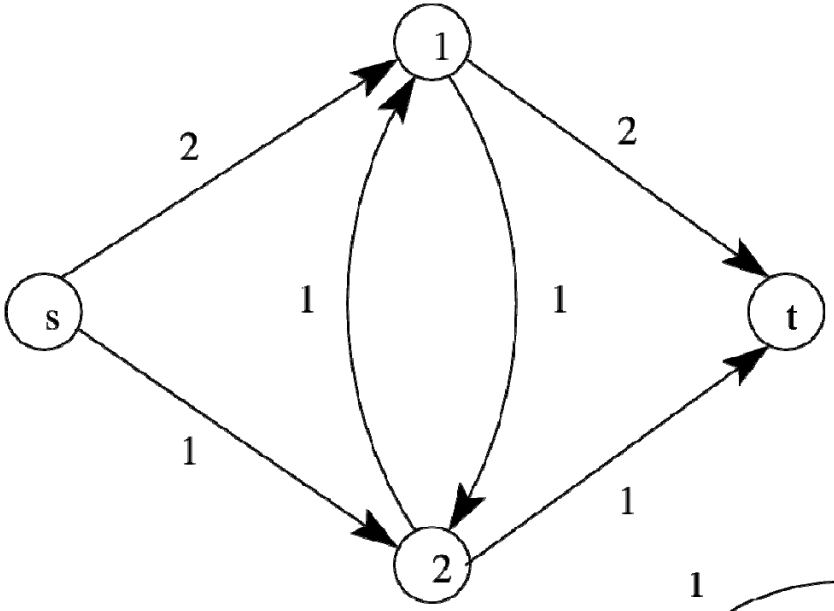
Outline

- Network flow
  - Max Flow (lab 2.6)
  - Min Cut (lab 2.7)
  - Min Cost Max Flow (lab 2.8)
- Matching problems
  - Maximum Cardinality Bipartite Matching (UVA 259)
  - Maximum Weighted Bipartite Matching
  - Maximum Cardinality/Weighted Matching
- Covering problems
  - Euler Path (lab 2.9)
  - Minimum Vertex Cover (Maximum Independent Set)
  - Minimum Path Cover in DAG

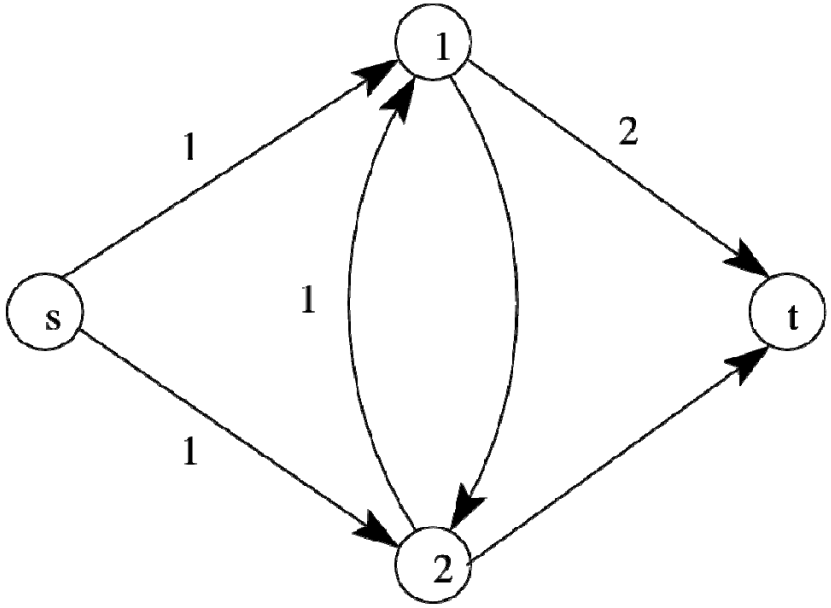
Network Flow

- A network is a directed graph G=(V,E) with a *source* vertex s∈V and a *sink* vertex t∈V. Each edge e=(v,w) from v to w has a defined *capacity*, denoted by u(e) or u(v,w). It is useful to also define capacity for any pair of vertices (v,w)∉E with u(v,w)=0.
- In a network flow problem, we assign a *flow* to each edge.
  - *Raw flow* is a function r(v,w) that satisfies the following properties:
    - **Conservation**: The total flow entering v must equal the total flow leaving v for all vertices except s and t, $\sum_{w \in V} r(v,w)=0$, for all $v \in V \setminus \{s,t\}$.
    - **Capacity constraint**: The flow along any edge must be positive and less than the capacity of that edge, $r(v,w) \leq u(v,w)$ for all $v,w \in V$.
  - *Net flow* is a function f(v,w) that also satisfies the following conditions:
    - **Skew symmetry**: f(v,w)=–f(w,v).
  - With a raw flow, we can have flows going both from v to w and flow going from w to v. In a net flow formulation however, we only keep track of the difference between these two flows f(v,w)=r(v,w)–r(w,v).
- The value of a flow f is defined as $|f|=\sum_{v \in V} f(s,v)$.

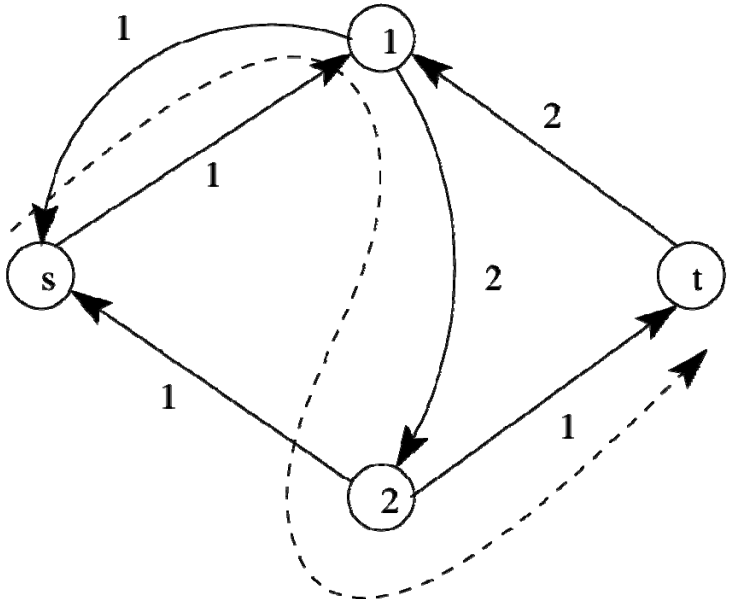
Network Flow – Example Network



Network

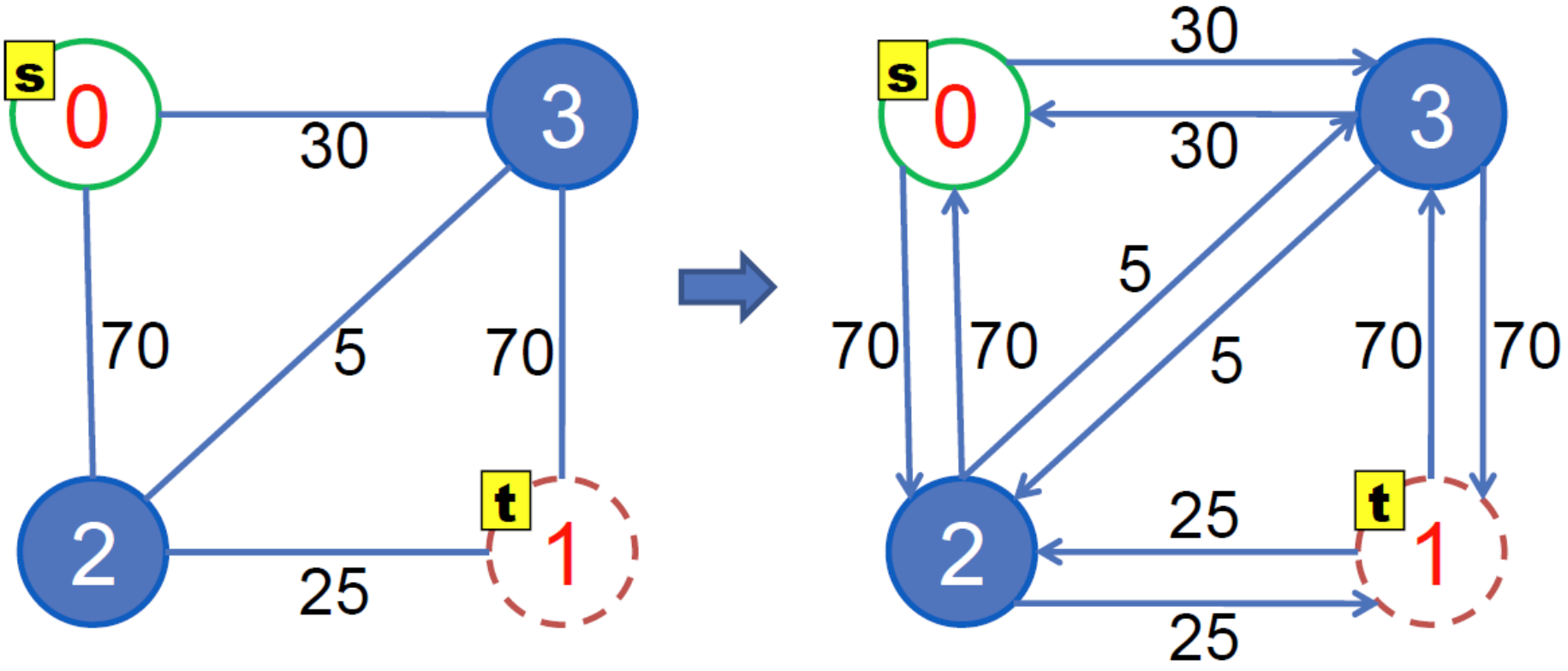


Raw flow

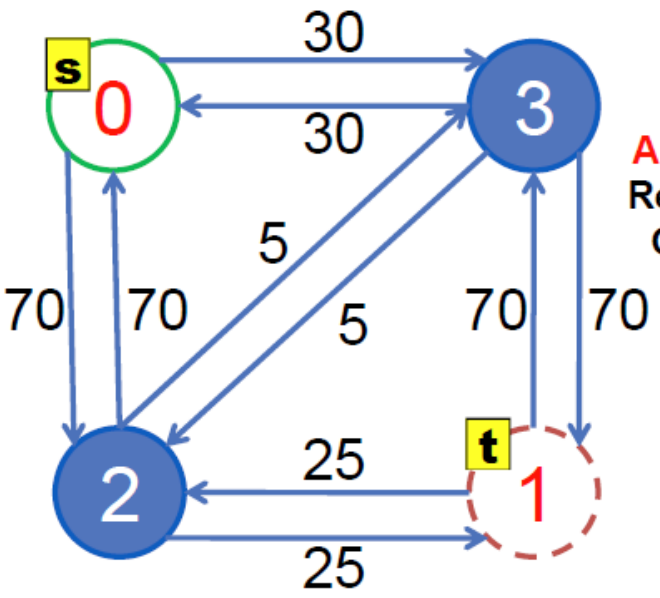


Residual graph and augmenting path

Network Flow – Example Maximum Flow



Network Flow – Example Maximum Flow



A. Initial Residual Graph

Network Flow – Scaling

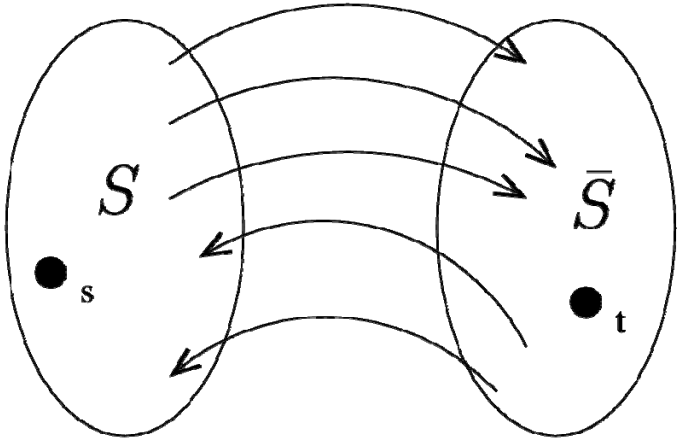
- We can also improve the running time of the Ford-Fulkerson algorithm by using a scaling algorithm. The idea is to reduce our max flow problem to the simple case, where all edge capacities are either 0 or 1 (Gabow in 1985 and Dinic in 1973):
  - Scale the problem down somehow by rounding off lower order 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 some max 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.
  - Increment some of the capacities. This restores the lower order bits that we truncated. Find augmenting paths in the residual network to re-maximize the flow.

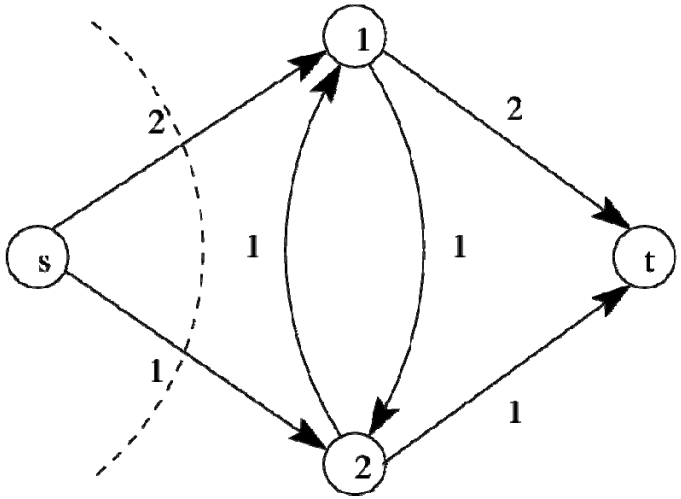
Maximum Flow Algorithms

- Ford-Fulkerson with DFS $O(|f| E)$
- Edmond-Karp (Ford-Fulkerson with BFS) $O(VE^2)$
- Dinic's $O(V^2E)$
- Push-relabel $O(V^3)$
- Binary blocking flow algorithm $O(min(V^{2/3}, E^{1/2}) E \log(V^2/E) \log(|f|))$

Minimum Cut

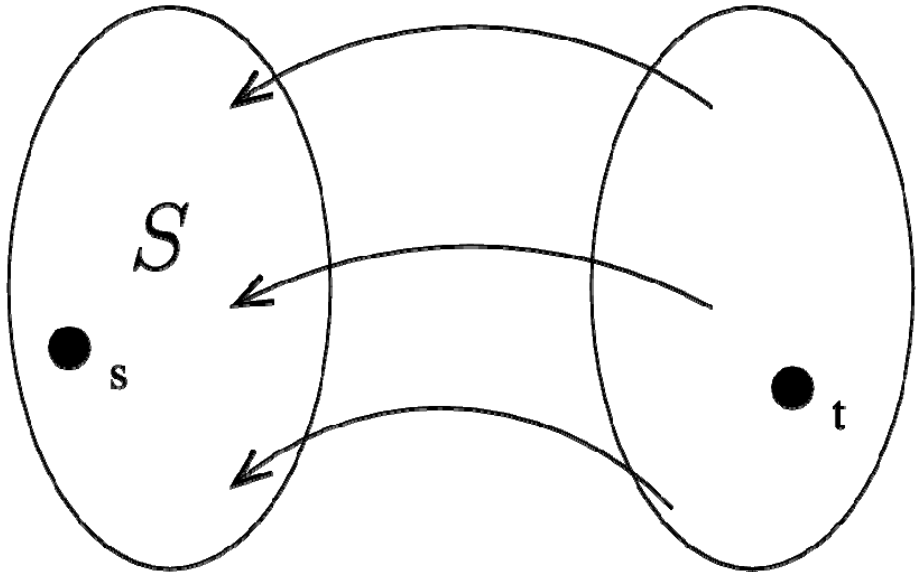
- An *s-t cut* of network G is a partition of the vertices V into 2 groups: S and $S^- = V \setminus S$ such that $s \in S$ and $t \in S^-$.
  - The net flow along cut (S, S^-) is defined as $f(S) = \sum_{v \in S} \sum_{w \in S^-} f(v,w)$.
  - The value (or capacity) of a cut is defined as $u(S) = \sum_{v \in S} \sum_{w \in S^-} u(v,w)$.
- For a flow network, we define a *minimum cut* to be a cut of the graph with minimum capacity.
- To find the minimum cut, compute the maximum flow and find the set of vertices reachable from s with positive edges in the residual graph, this is the set S.





Max-Flow Min-Cut Theorem

- In a flow network G, the following conditions are equivalent:
  - A flow f is a maximum flow.
  - The residual network G_f has no augmenting paths.
  - $|f|=u(S)$ for some cut S.
- These conditions imply that the value of the maximum flow is equal to the value of the minimum s-t cut: $\max_f |f|=\min_S u(S)$, where f is a flow and S is an s-t cut.



Minimum Cost Maximum Flow

- Extend the definition of a network flow with a cost per unit of flow on each edge: $c(v,w) \in R$, where $(v,w) \in E$.
- The cost of a flow f is defined as: $c(f) = \sum_{e \in E} f(e) \cdot c(e)$
- A minimum cost maximum flow of a network $G=(V,E)$ is a maximum flow with the smallest possible cost.
  - Note that costs can be negative.
  - It's clear that minimum cost maximum flow generalizes max-flow, if assign a cost of 0 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.
  - Note that edges in the residual graph of a network need to have their costs determined carefully. Consider an edge (v,w) with capacity u(v,w), cost per unit flow c(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 $u(v,w)-f(v,w)$ and cost c(v,w), and second edge is (w,v) with capacity f(v,w) and cost $-c(v,w)$.
- A flow is **optimal (min-cost)** iff there are no negative cost cycles in the residual network.

Network Flow Variants

- Multi-source, multi-sink max flow
  - 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.
- Min-Cost Circulation
  - Equivalent to min-cost max-flow (simply disconnect the source and sink)
- Maximum Independent and Edge-Disjoint Paths

Euler Path

- Hierholzer's algorithm
  - Choose any starting vertex v, and follow a trail of edges from that vertex until returning to v. It is not possible to get stuck at any vertex other than v, because the even degree of all vertices ensures that, when the trail enters another vertex w there must be an unused edge leaving w. The tour formed in this way is a closed tour, but may not cover all the vertices and edges of the initial graph.
  - As long as there exists a vertex v that belongs to the current tour but that has adjacent edges not part of the tour, start another trail from v, following unused edges until returning to v, and join the tour formed in this way to the previous tour.
- By using a data structure such as a doubly linked list to maintain the set of unused edges incident to each vertex, to maintain the list of vertices on the current tour that have unused edges, and to maintain the tour itself, the individual operations of the algorithm (finding unused edges exiting each vertex, finding a new starting vertex for a tour, and connecting two tours that share a vertex) may be performed in constant time each, so the overall algorithm takes linear time.

Summary

- Network flow
  - Max Flow (lab 2.6)
  - Min Cut (lab 2.7)
  - Min Cost Max Flow (lab 2.8)
- Matching problems
  - Maximum Cardinality Bipartite Matching (UVA 259)
  - Maximum Weighted Bipartite Matching
  - Maximum Cardinality/Weighted Matching
- Covering problems
  - Euler Path (lab 2.9)
  - Minimum Vertex Cover (Maximum Independent Set)
  - Minimum Path Cover in DAG