CMPSCI 575/MATH 513

Combinatorics and Graph Theory

Lecture #13: Flows in Networks
(Tucker Section 4.3)
David Mix Barrington
5 October 2016

Flows in Networks

- Definitions: Flows and Cuts
- Min Cuts and Max Flows
- A Faulty Algorithm
- The Augmenting Flow Algorithm
- Example of Bad Flow Choice
- Max Flows in Polynomial Time
- Applications of Network Flow

Definitions: Flows and Cuts

- Today we will look at **flows** in a **network**.
- A network is a directed graph with a **source node** a and a **sink node** z. The labels on the directed edges represent **capacities**, the maximum amount of "stuff" that can be carried on the edge.
- A flow is an assignment of a non-negative number (an amount of stuff) to each edge, that is bounded by the capacity.

Definitions: Flows and Cuts

- A flow must satisfy some rules. There is zero flow into the source or out of the sink. And at every other node, the total flow in must equal the total flow out, so the net flow is zero.
- The value of a flow is the total out of a or into z.
- A **cut** is a partition of the nodes into two sets, a set P containing a and a set Q containing z. The flow across the cut is the sum of flows on edges from P to Q, minus the sum of flows on edges from Q to P.

Capacity of a Cut

- Given any cut (P, Q), the **capacity of the cut** is the sum of the capacities of every edge with one endpoint in P and the other in Q.
- It is easy to see that the net flow across any cut in a flow is always equal to the value of the flow. This is because the net flow at any middle node is zero.
- Thus no flow can have a value exceeding the minimum capacity of any cut.

Min Cuts and Max Flows

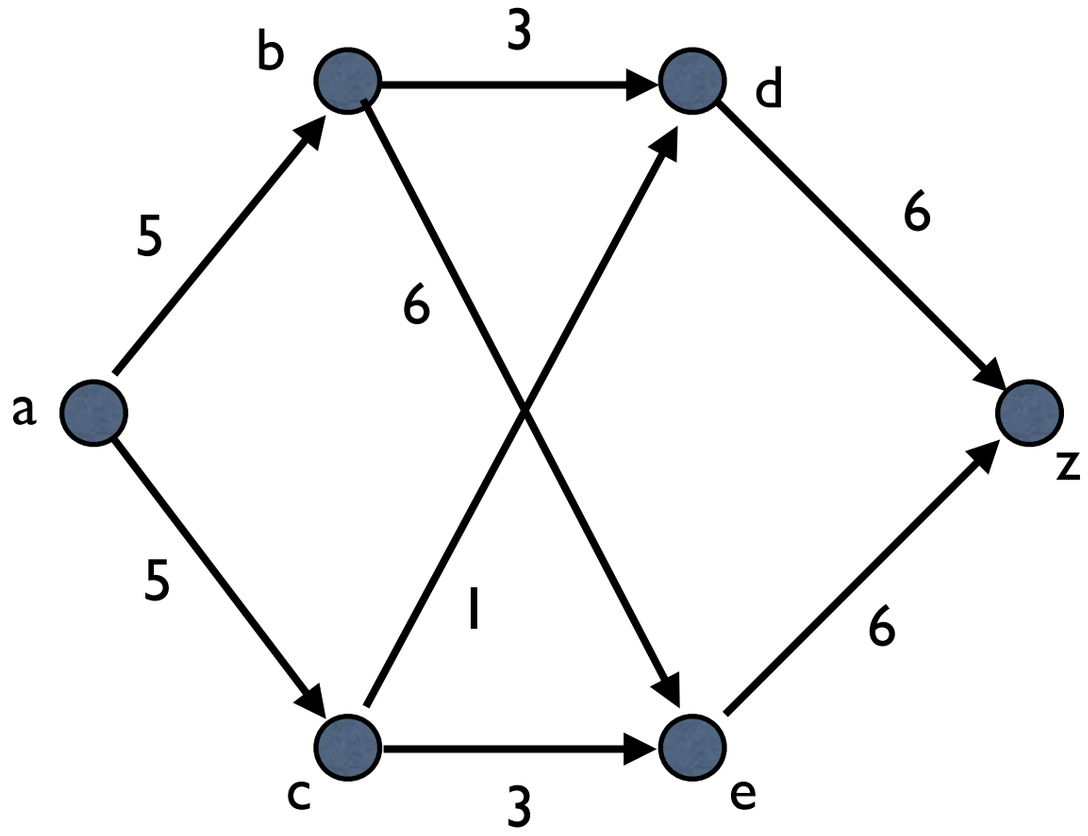
- The main result of this lecture is that a flow meeting the capacity of the minimum cut can always be achieved. This will be the maximum possible value of a flow through that network.
- We say that an edge is **saturated** by a flow if the flow through it equals the capacity. If an edge is not saturated, it has a **slack** equal to the additional flow that it could accept.

Faulty Flow Building

- Here is a simple idea to build a maximum flow. Look at the slack edges, find a path of slack edges from a to z, find the maximum positive flow possible along that path, and add it to the flow. If there is no such path of slack edges, the edges reachable from a by slack edges form the set P of a cut, which is saturated.
- Unfortunately this simple idea is **wrong.**

Faulty Flow Example

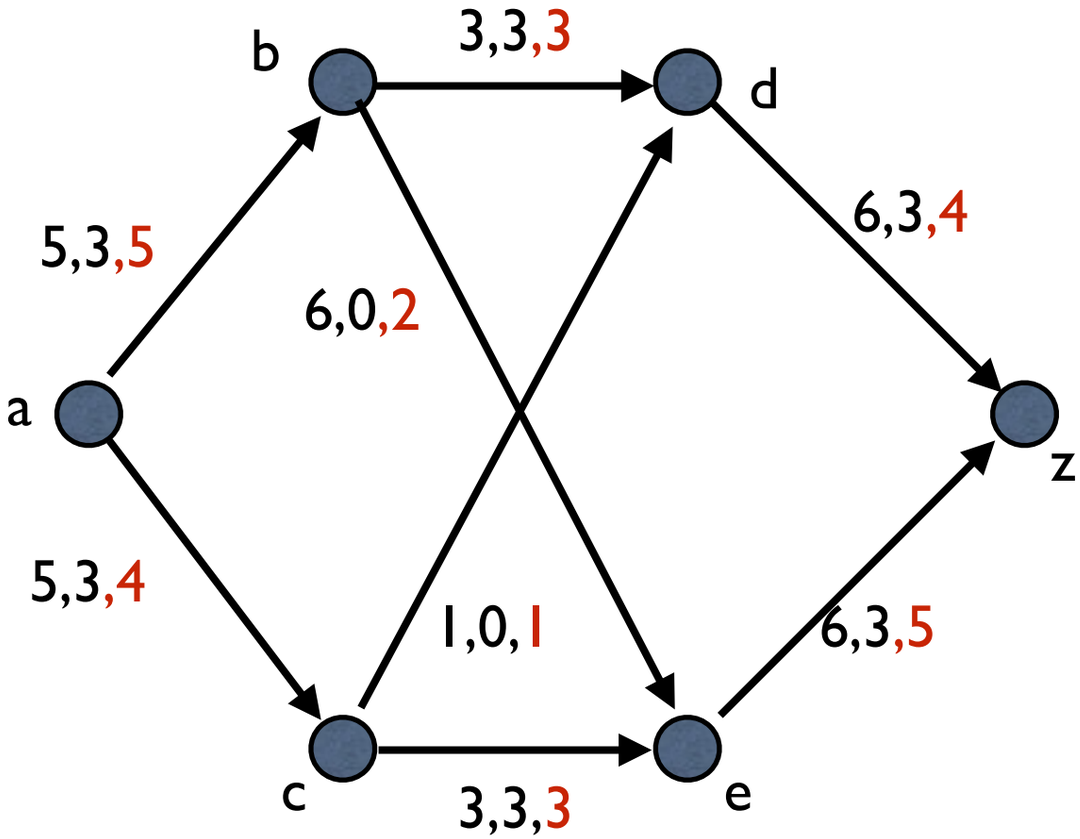
- Here is a network with capacities as shown. The cut with just a has capacity 10, so we can't get more flow than that.



- We can put 3 on a-b-d-z, then 3 on a-c-e-z.

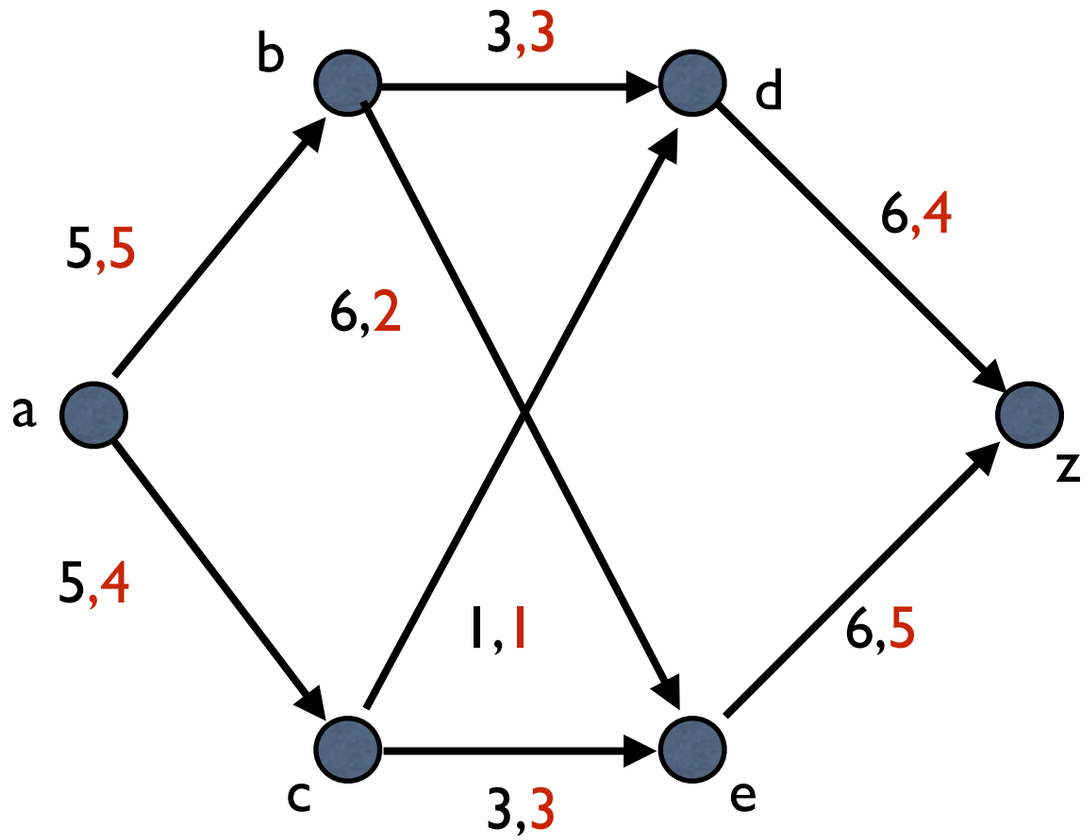
Faulty Flow Example

- We put this flow as the second label.
- Now the path a-b-e-z can take 2, leaving a-c-d-z as the only remaining path of slack edges, with minimum slack 1.
- This gives the flow of value 9 in red.



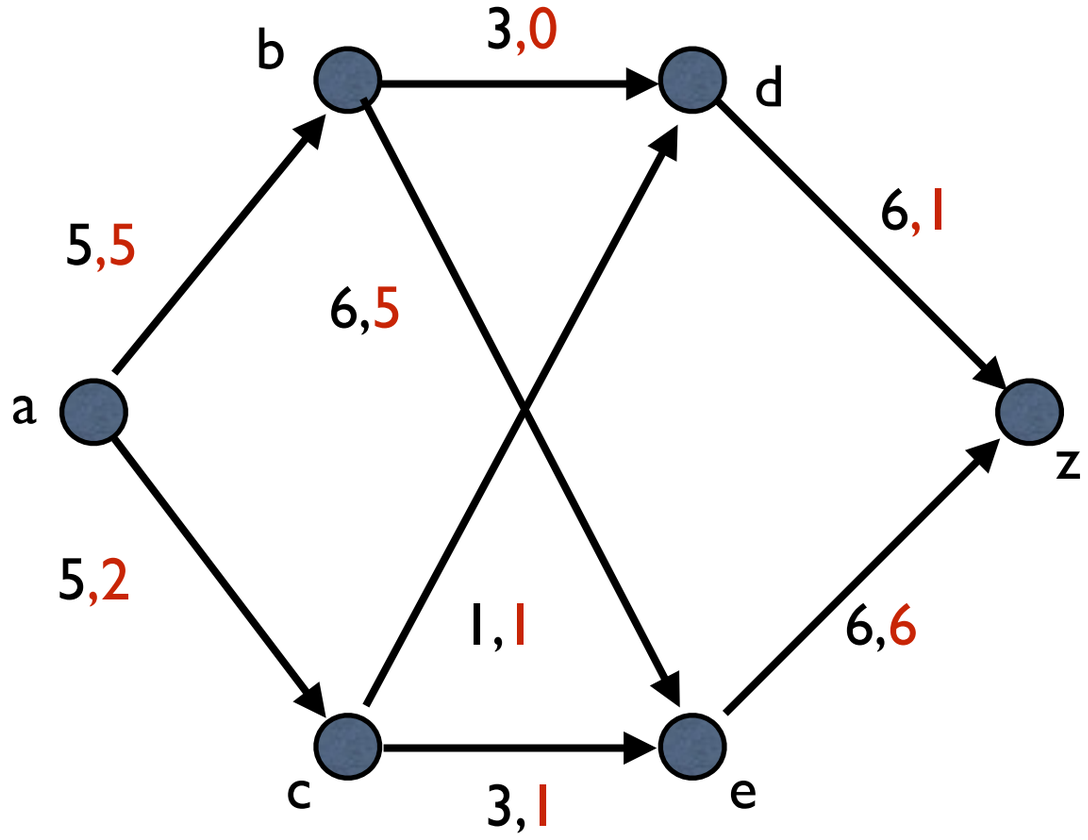
Faulty Flow Example

- There are slack edges with this flow: (a,c), (b,e), (d,z), and (e,z), but no slack path from a to z.
- The cut {a, c} has capacity 9, so this is a maximum flow.



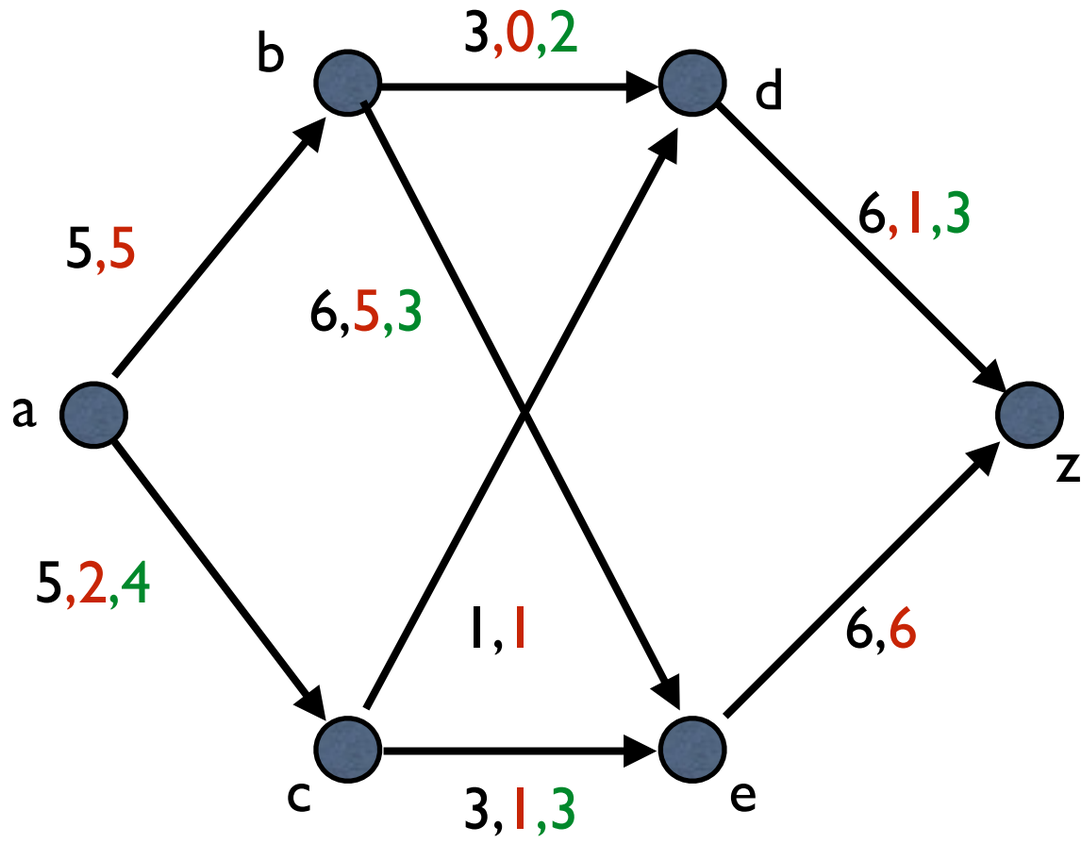
Faulty Flow Example

- Suppose instead our first flows were 5 on a-b-e-z and 1 on a-c-d-z.
- This leaves a-c-e-z as the only path of slack edges, so we put 1 there for a value of 7.
- Now no edges out of {a,c,e} have slack, but we don't have max flow.



Faulty Flow Example

- The capacity of cut {a,c,e} is 12, but we are losing 5 to edge (b, e).
- Consider a virtual path a-c-e-b-d-z, where we go backward along (b,e) by subtracting 2 from the flow and add 2 on the other edges. This gives the max flow in green, with value 9.



The Augmenting Flow Algorithm

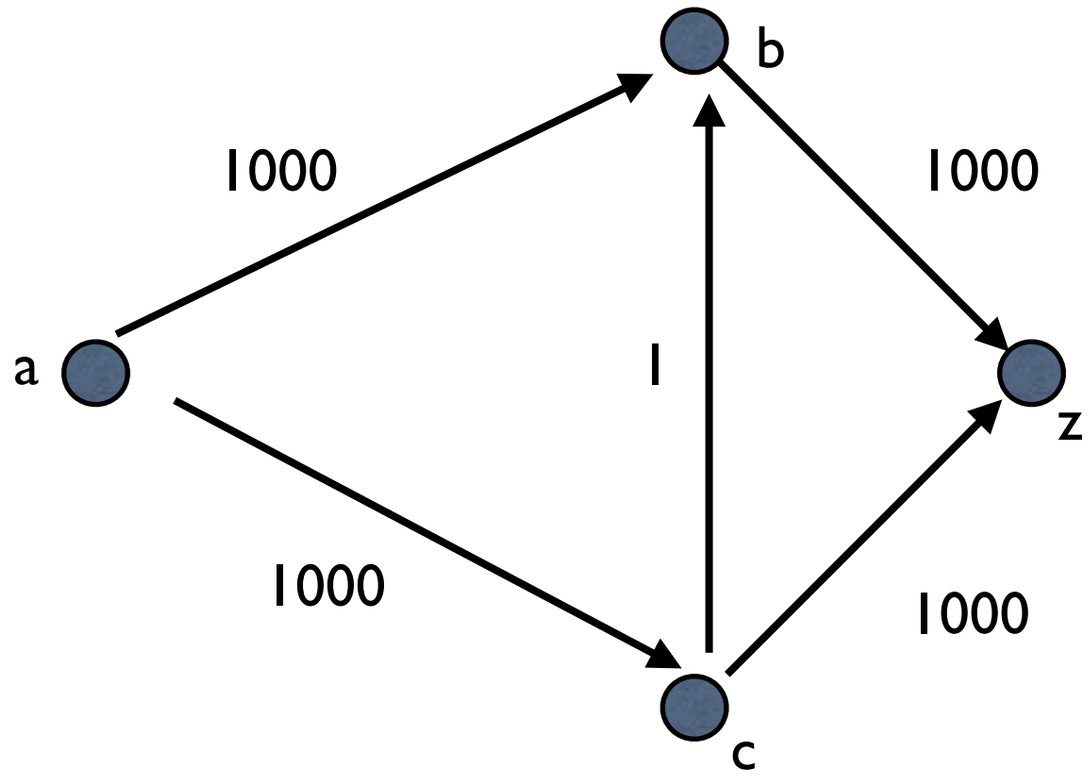
- This gives us our first correct algorithm to find the max flow in a network.
- We define a **chain** to be a sequence of edges that form a path if some of them are reversed.
- An **augmenting chain** for a flow in a network consists of slack edges going forward and edges with positive flow going backward.

The Augmenting Flow Algorithm

- Adding an augmenting chain to a flow gives a new flow with bigger capacity.
- The **augmenting flow algorithm** repeatedly searches for such a chain, adding it to the flow, until there is none available.
- Since we are working with integers, this can't go on forever. If we are stuck, the nodes reachable from a by partial chains form a cut, and the flow across that cut is maximum.

Example of Bad Flow Choice

- Choosing augmenting chains will get us to a maximum flow, but not necessarily quickly.
- In this graph, we can get to the maximum flow by 2000 steps of adding 1 flow each, first using a-c-b-z and then a-b-c-z.



Max Flows in Polynomial Time

- It turns out that a simple idea solves this problem: when searching for an augmenting chain, use BFS and thus get a chain with the minimum possible number of edges.
- It can be shown (usually in CS 311) that this method can saturate a given edge only O(n) times, meaning O(ne) augmenting chains and thus $O(ne^2)$ time, independent of the edge weights.

Applications of Network Flow

- If we want to find a maximum flow in an *undirected* network, we just convert each edge to two directed edges with the same capacity, and delete edges into a or out of z.

- If we want to find the maximum number of **edge-disjoint paths** in a graph, we make a network by adding new nodes a and z connected to each old node by an edge of capacity 1. We know the flow found by our algorithm will have integer flow on each edge.

Applications of Network Flow

- Finally, we can apply network flow to what would have been the subject of Lecture #14, **maximum matchings** in bipartite graphs.
- Given a graph where all edges between nodes in A and nodes in B, we direct each edge from a to b, give it capacity 1, then add a new node a with edges to each node in A and a new node z with edges from each node in B.
- A maximum flow in this network is a maximum matching in the original graph.

Hall's Theorem

- I can't leave the subject of matchings without mentioning the beautiful theorem of Hall.
- In a bipartite graph with vertex sets A and B of size n, for any set $S \subseteq A$, let $N(S) \subseteq B$ be the set of nodes with neighbors in A.
- If $|N(S)| < |S|$, we have a bottleneck and clearly a perfect matching is impossible.
- Hall's Theorem is the converse, that if there are no bottlenecks, a perfect matching exists.

Hall's Theorem

- We can prove this with network flow. The set a ∪ S ∪ N(S) always forms a cut in our network, of capacity at most n - |S| + |N(S)|.
- There's a nice elementary proof by induction on n. Let $x \in A$ and pick $y \in N(\{x\})$. If removing x and y does creates no bottleneck, we win. Otherwise let S of size k be the bottleneck set in the new graph, and note that |N(S)| must be k-1 because y has to have been in N(S) in the old graph.

Hall's Theorem

- By the IH, S has a perfect matching with the old N(S), which includes y. We want to apply the IH as well to S^C, to show that it has a perfect matching with $N(S)^C$.
- Let T be any subset of S^C. We want to show that $|T| = |N(T) \cap N(S)^C|$. But we know that $N(T \cup S)$ has the same size as $T \cup S$, and the elements of $N(T \cup S)$ - N(S) must have neighbors in T as they don't have them in S.