CMSC 451: Network Flows

Slides By: Carl Kingsford

Department of Computer Science
University of Maryland, College Park

Based on Sections 7.1&7.2 of *Algorithm Design* by Kleinberg & Tardos.

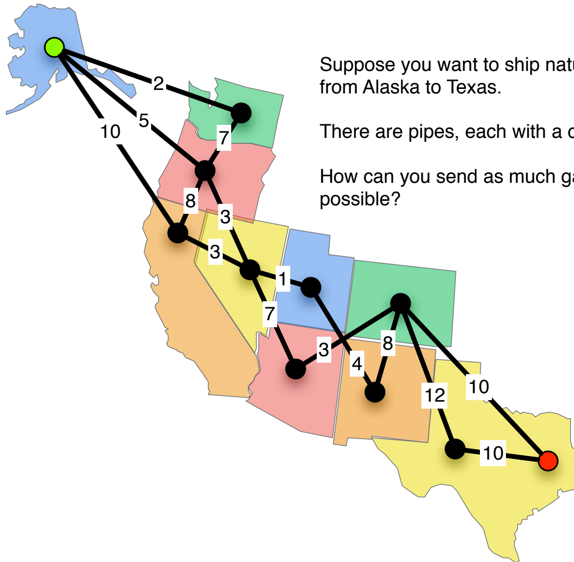
Network Flows

- Our 4th major algorithm design technique (greedy, divide-and-conquer, and dynamic programming are the others).

- A little different than the others: we'll see an algorithm for one problem (and minor variants) that is **so useful** that we can apply to to many practical problems.

- Called **network flow**.

Network flow problem, e.g.



Suppose you want to ship natural gas from Alaska to Texas.

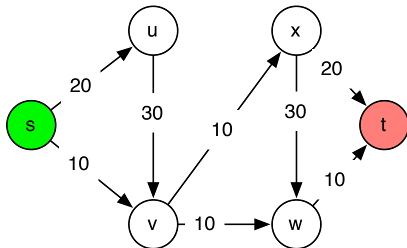
There are pipes, each with a capacity.

How can you send as much gas as possible?

Flow Network

A **flow network** is a connected, directed graph $G = (V, E)$.

- Each edge e has a non-negative, integer **capacity** c_e.
- A single **source** $s \in V$.
- A single **sink** $t \in V$.
- No edge enters the source and no edge leaves the sink.



Assumptions

To repeat, we make these assumptions about the network:

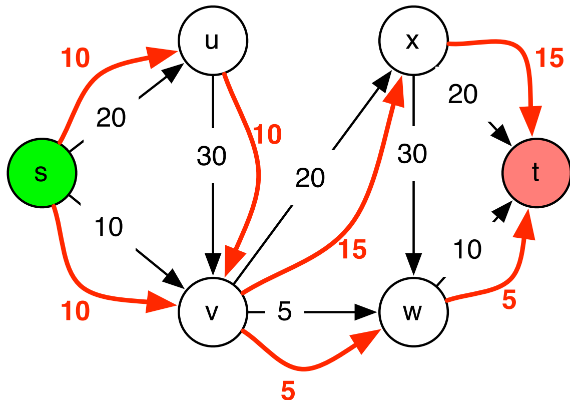
1. Capacities are integers.
2. Every node has one edge adjacent to it.
3. No edge enters the source and no edge leaves the sink.

These assumptions can all be removed.

Flow

Def. An **s-t flow** is a function $f : E \to \mathbb{R}^{\geq 0}$ that assigns a real number to each edge.

Intuitively, $f(e)$ is the amount of material carried on the edge e.



Flow constraints

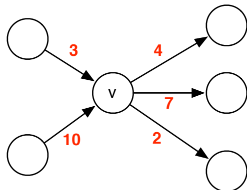
Constraints on f:

1. $0 \le f(e) \le c_e$ for each edge e. (capacity constraints)

2. For each node v except s and t, we have:

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ leaving } v} f(e)\,.$$

(balance constraints: whatever flows in, must flow out).



Notation

The **value** of flow f is:

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

This is the amount of material that s is able to send out.

Notation:

- $f^{\text{in}}(v) = \sum_{e \text{ into } v} f(e)$
- $f^{\text{out}}(v) = \sum_{e \text{ leaving } v} f(e)$

Balance constraints becomes: $f^{\text{in}}(v) = f^{\text{out}}(v)$ for all $v \notin \{s, t\}$

Maximum Flow Problem

Definition (Value)

The value $v(f)$ of a flow f is $f^{\mathrm{out}}(s)$.

That is: it is the amount of material that leaves s.

Maximum Flow Problem

Given a flow network G, find a flow f of maximum possible value.

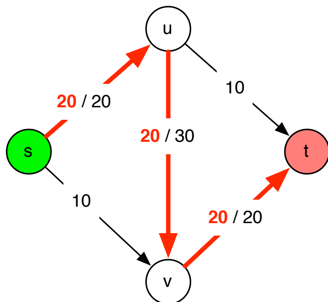
A Greedy Start

A Greedy Start:

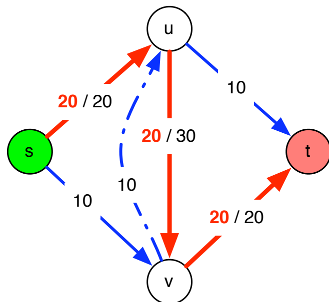
1. Suppose we let $f(e) = 0$ for all edges (no flow anywhere).
2. Choose some $s - t$ path and "push" flow along it up to the capacities. Repeat.
3. When we get stuck, we can erase some flow along certain edges.

How do we make this more precise?

Example



After 1 path, we've allocated 20 units



Want to send the blue path

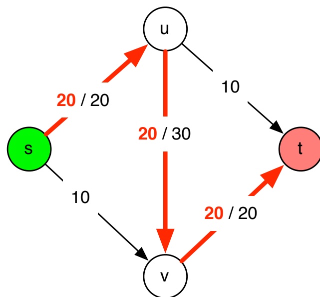
Residual Graph

We define a **residual graph** G_f. G_f depends on some flow f:

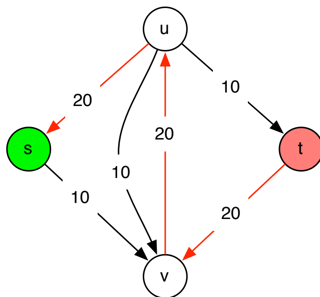
1. G_f contains the same nodes as G.
2. **Forward edges**: For each edge $e = (u, v)$ of G for which $f(e) < c_e$, include an edge $e' = (u, v)$ in G_f with capacity $c_e - f(e)$.
3. **Backward edges:** For each edge $e = (u, v)$ in G with $f(e) > 0$, we include an edge $e' = (v, u)$ in G_f with capacity $f(e)$.

Residual Graph, e.g.

- If $f(e) < c_e$, add edge e to G_f with capacity $c_e - f(e)$.
  (remaining capacity left)
- If $f(u, v) > 0$, add reverse edge (v, u) with capacity $f(e)$.
  (can erase up to $f(e)$ capacity)



With Flow f



Residual Graph G_f

Augmenting Paths

- Let P be an $s - t$ path in the residual graph G_f.
- Let $\mathrm{bottleneck}(P, f)$ be the smallest capacity in G_f on any edge of P.
- If $\mathrm{bottleneck}(P, f) > 0$ then we can increase the flow by sending $\mathrm{bottleneck}(P, f)$ along the path P.

Augmenting Paths, 2

If bottleneck$(P, f) > 0$ then we can increase the flow by sending bottleneck(P, f) along the path P:

```
augment(f, P):
   b = bottleneck(P,f)
   For each edge (u,v) ∈ P:
      If e = (u,v) is a forward edge:
         Increase f(e) in G by b   //add some flow
      Else:
         e’ = (v,u)
         Decrease f(e’) in G by b  //erase some flow
      EndIf
   EndFor
   Return f
```

Ford-Fulkerson Algorithm

```
MaxFlow(G):
  // initialize:
  Set f[e] = 0 for all e in G

  // while there is an s-t path in Gf:
  While P = FindPath(s,t, Residual(G,f)) != None:
     f = augment(f, P)
     UpdateResidual(G, f)
  EndWhile
  Return f
```

After augment, we still have a flow

After $f' = \texttt{augment}(P, f)$, we still have a flow:

Capacity constraints: Let e be an edge on P:

- if e is forward edge, it has capacity $c_e - f(e)$. Therefore,

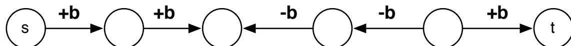
  $$f'(e) = f(e) + \mathbf{bottleneck}(P, f) \leq f(e) + c_e - f(e) \leq c_e$$

- if e is a backward edge, it has capacity $f(e)$. Therefore,

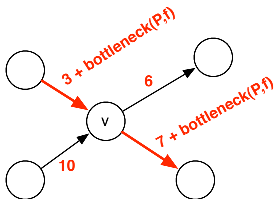
  $$f'(e) = f(e) - \mathbf{bottleneck}(P, f) \geq f(e) - f(e) = 0$$

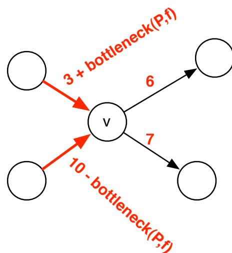
Still have flow, 2

Balance constraints: An s-t path in G_f corresponds to some set of edges in G:



In other pictures,





Running Time

1. At every step, the flow values $f(e)$ are integers.

2. At every step we increase the amount of flow $v(f)$ sent by at least 1 unit.

3. We can never send more than $C := \sum_{e \text{ leaving } s} c_e$.

Theorem

The Ford-Fulkerson algorithm terminates in C iterations of the `While` loop.

Running Time

1. At every step, the flow values $f(e)$ are integers. Start with ints and always add or subtract ints
2. At every step we increase the amount of flow $v(f)$ sent by at least 1 unit.
3. We can never send more than $C := \sum_{e \text{ leaving } s} c_e$.

Theorem

The Ford-Fulkerson algorithm terminates in C iterations of the `While` loop.

Time in the While loop

1. If G has m edges, G_f has $\leq 2m$ edges.
2. Can find an $s - t$ path in G_f in time $O(m + n)$ time with DFS or BFS.
3. Since $m \geq n/2$ (every node is adjacent to some edge), $O(m + n) = O(m)$.

Theorem

The Ford-Fulkerson algorithm runs in $O(mC)$ time.

Caveats, etc.

Note this is **pseudo-polynomial** because it depends on the size of the integers in the input.

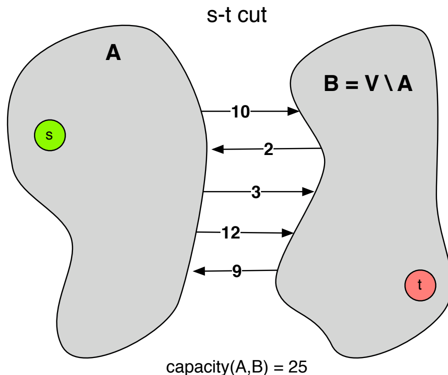
You can remove this with slightly different algorithms. E.g.:

- $O(nm^2)$: Edmonds-Karp algorithm (use BFS to find the augmenting path)
- $O(m^2 \log C)$ (see section 7.3), or
- $O(n^2 m)$ or $O(n^3)$ (see section 7.4).

Proof of correctness

How do we know the flow is maximum?

Cuts and Cut Capacity



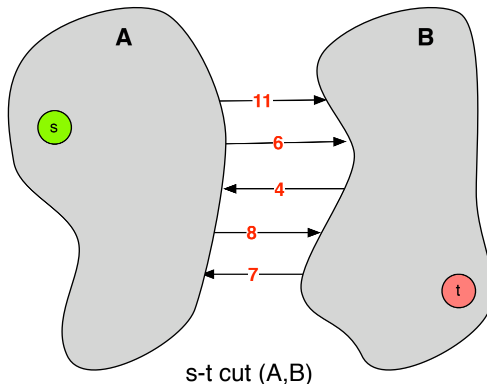
Definition

The **capacity**(A, B) of an s-t cut (A, B) is the sum of the capacities of edges leaving A.

A Cut Theorem

Theorem

Let f be an s-t flow and (A, B) be an s-t cut. Then $v(f) = f^{out}(A) - f^{in}(A)$.



s-t cut (A,B)

Another cut theorem

Theorem

Let f be any s-t flow and (A, B) be any s-t cut.
Then $v(f) \leq capacity(A, B)$.

Proof.

$$\begin{aligned} v(f) &= f^{\text{out}}(A) - f^{\text{in}}(A) && \text{prev. thm} \\ &\leq f^{\text{out}}(A) && f^{\text{in}}(A) \text{ is } \geq 0 \\ &= \sum_{e \text{ leaving } A} f(e) && \text{by definition} \\ &\leq \sum_{e \text{ leaving } A} c_e && \text{by capacity constraints} \\ &= \text{capacity}(A, B) && \text{by definition} \end{aligned}$$

Cuts Constrain Flows

This theorem:

Theorem

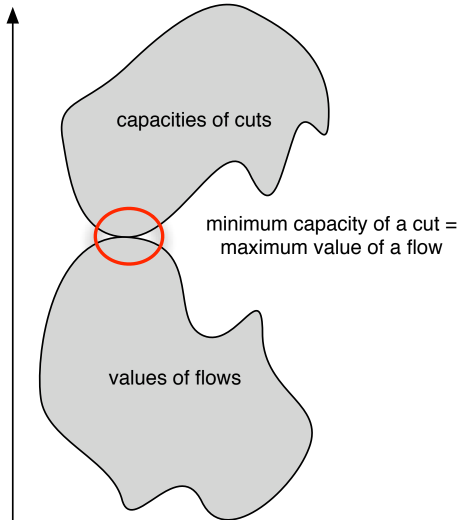
Let f be any s-t flow and (A, B) be any s-t cut.
Then $v(f) \leq \mathit{capacity}(A, B)$.

Says that any cut is bigger than any flow.

Therefore, cuts constrain flows. The minimum capacity cut constrains the maximum flow the most.

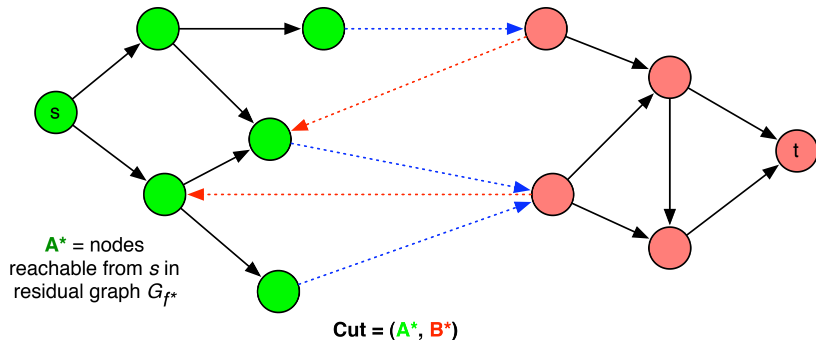
In fact, the capacity of the minimum cut always **equals** the maximum flow value.

Max-Flow = Min-Cut



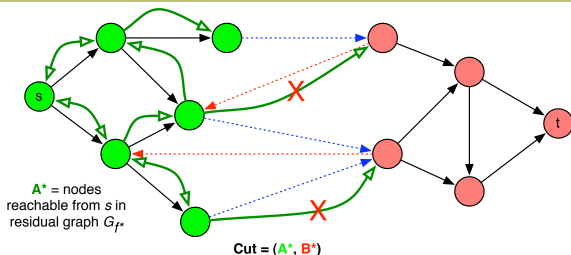
Max-Flow = Min-Cut, 2

Let f^* be the flow returned by our algorithm.
Look at G_{f^*} but define a cut in G:



Blue edges must be saturated.
Red edges must have 0 flow $\implies v(f^*) = \textbf{capacity}(A^*, B^*)$.

Max-Flow = Min-Cut, 3



- (A^*, B^*) is an s-t cut because there is no path from s to t in the residual graph G_{f^*}.
- Edges (u, v) from A^* to B^* must be saturated — otherwise there would be a forward edge (u, v) in G_{f^*} and v would be part of A^*.
- Edges (v, u) from B^* to A^* must be empty — otherwise there would be a backedge (u, v) in G_{f^*} and v would be part of A^*.

Max-Flow = Min-Cut, 4

Therefore,

- $v(f^*) = \textbf{capacity}(A^*, B^*)$.
- No flow can have value bigger than $\textbf{capacity}(A^*, B^*)$.
- So, f^* must be an maximum flow.
- And (A^*, B^*) has to be a minimum-capacity cut.

Theorem (Max-flow = Min-cut)

The value of the maximum flow in any flow graph is **equal to** *the capacity of the minimum cut.*

Finding the Min-capacity Cut

Our proof that maximum flow = minimum cut can be used to actually **find** the minimum capacity cut:

1. Find the maximum flow f^*.
2. Construct the residual graph G_{f^*} for f^*.
3. Do a BFS to find the nodes reachable from s in G_{f^*}. Let the set of these nodes be called A^*.
4. Let B^* be all other nodes.
5. Return (A^*, B^*) as the minimum capacity cut.

To Summarize

Summary:

- Ford-Fulkerson algorithm can find max flow in $O(mC)$ time.
- **Algorithm idea:** Send flow along some path with capacity left, possibly "erasing" some flow we've already sent. Use residual graph to keep track of remaining capacities and flow we've already sent.
- We can eliminate C to get a true polynomial algorithm by using BFS to find our augmenting paths.
- All cuts have capacity $\geq$ the value of all flows.
- Know the flow is maximum because its value equals the capacity of some cut.