CS 312: Algorithms

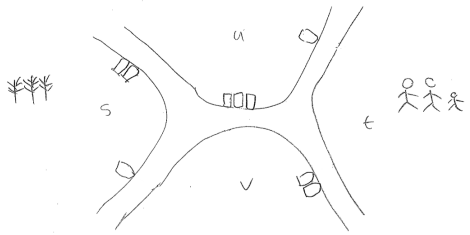
Lecture 19: Network Flows

Dan Sheldon

Mount Holyoke College

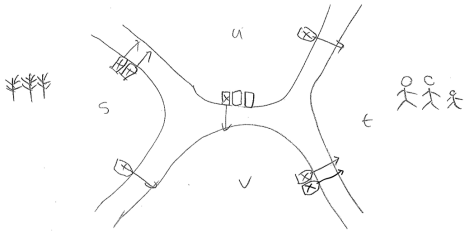
Last Compiled: November 14, 2018

A Puzzle

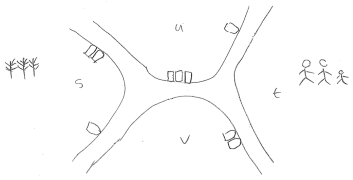


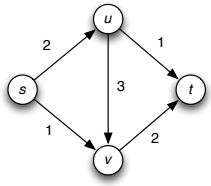
How many loads of grain can you ship from s to t? Which boats are used?

A Puzzle

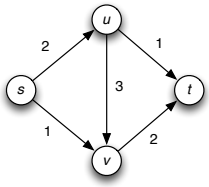


Flow Network





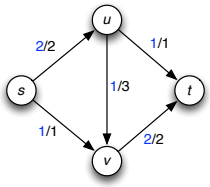
Max-Flow Problem



Problem input is a **flow network**

- Directed graph
- Source node s
- Target node or *sink* t
- Edge capacities $c(e) \geq 0$

Solution: A Flow



A **network flow** is an assignment of values $f(e)$ to each edge e, which satisfy:

- Capacity constraints: $0 \leq f(e) \leq c(e)$ for all e
- Flow conservation:

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

  for all $v \notin \{s, t\}$.
- Value $v(f)$ of flow f = total flow on edges leaving source
- **Max flow problem**: find a flow of maximum value

Algorithm Design Techniques

- Greedy
- Divide and Conquer
- Dynamic Programming
- Network Flows

Network Flow

- Previous topics (Greedy, Divide-and-Conquer, Dynamic Programming) were design techniques
- Network flow relates to a specific class of problems with many applications

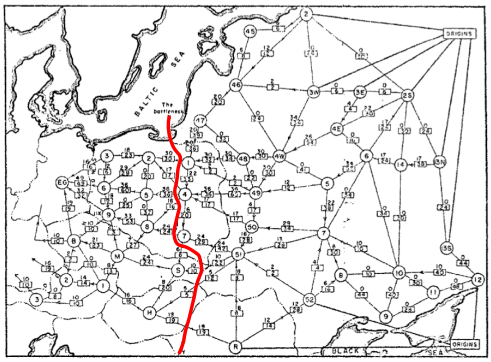
- Direct applications: commodities in networks
  - transporting food on the rail network
  - packets on the internet
  - gas through pipes
- Indirect applications:
  - Matching in graphs
  - Airline scheduling
  - Baseball elimination

Plan: design and analyze algorithms for max-flow problem, then apply to solve other problems

First, a Story About Flow and Cuts

Key theme: flows in a network are intimately related to cuts

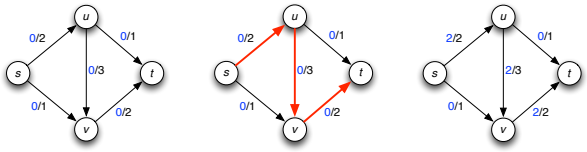
Soviet rail network in 1955



On the history of the transportation and maximum flow problems. Alexander Schrijver, Math Programming, 2002.

Designing a Max-Flow Algorithm

First idea: initialize to zero flow and then repeatedly "augment" flow on paths from s to t until we can no longer do so.

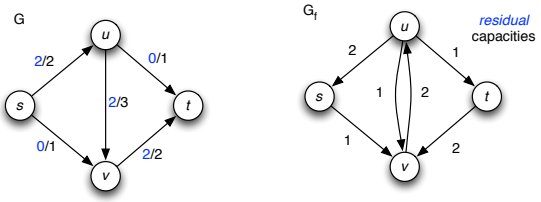


Problem: we are now stuck. All paths from s to t have a saturated edge.

We would like to "augment" $s \xrightarrow{+1} v \xleftarrow{-1} u \xrightarrow{+1} t$, but this is not a real $s \to t$ path. How can we identify such an opportunity?

Residual Graph

The residual graph G_f identifies opportunities to increase flow on edges with leftover capacity, or decrease flow on edges already carrying flow:



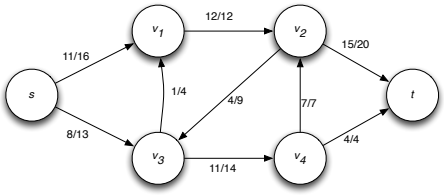
For each original edge $e = (u, v)$ in G, it has:

- A forward edge $e = (u, v)$ with *residual capacity* $c(e) - f(e)$
- A reverse edge $e' = (v, u)$ with *residual capacity* $f(e)$

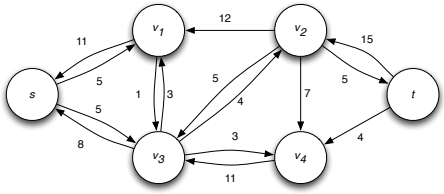
Edges with zero residual capacity are omitted

Exercise: draw the residual graph

G

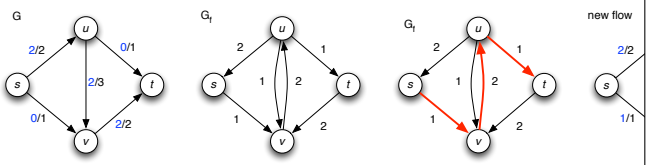


G_f



Augment Operation

Revised Idea: use paths in the *residual* graph to augment flow



- $P = s \to v \to u \to t$ has "bottleneck capacity" 1
  - bottleneck capacity $=$ smallest residual capacity of any edge in P
- Increase flow for forward edges, decrease for backward edges.
- Augment $s \xrightarrow{+1} v \xleftarrow{-1} u \xrightarrow{+1} t$

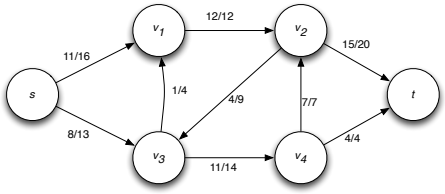
Augment Operation

Revised Idea: use paths in the *residual* graph to augment flow

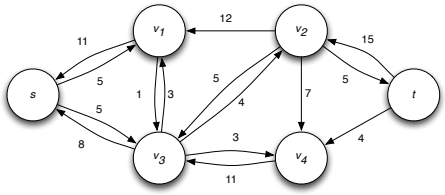
Augment(f, P)
  Let $b =$ bottleneck(P, f) ▷ least residual capacity in P
  for each edge e in P **do**
    if e is a forward edge **then**
      $f(e) = f(e) + b$ ▷ increase flow on forward edges
    else e is a backward edge
      Let e' be opposite edge in G
      $f(e') = f(e') - b$ ▷ decrease flow on backward edges
    end if
  end for

Augment Example

G

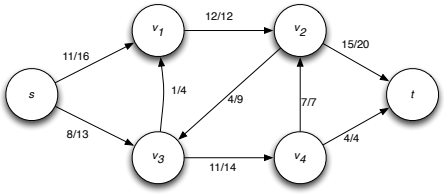


G_f

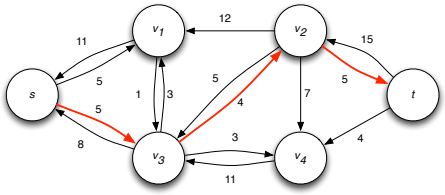


Augmenting Path

G

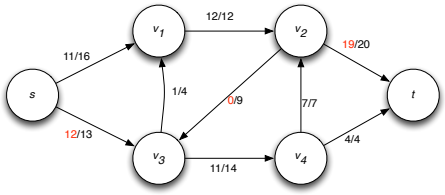


G_f

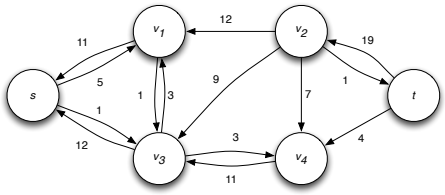


New Flow

G



G_f



Ford-Fulkerson Algorithm

Repeatedly find augmenting paths in the residual graph and use them to augment flow!

Ford-Fulkerson(G, s, t)
  ▷ Initially, no flow
  Initialize $f(e) = 0$ for all edges e
  Initialize $G_f = G$
  ▷ Augment flow as long as it is possible
  while there exists an s-t path P in G_f **do**
    $f =$ Augment(f, P)
    update G_f
  end while
  return f

Ford-Fulkerson Example

See Pearson slides

Ford-Fulkerson Analysis

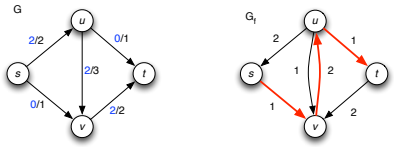
- Step 1: argue that F-F returns a flow
- Step 2: analyze termination and running time
- Step 3: argue that F-F returns a maximum flow

Step 1: F-F returns a flow

Claim: If f is a flow then $f' = \text{Augment}(f, P)$ is also a flow.

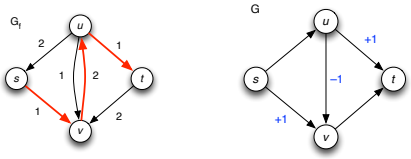
Proof idea. Verify two conditions for f' to be a flow: capacity and flow conservation.

Capacity



- Suppose original edge is $e = (u, v)$
- If e appears in P as a forward edge ($u \xrightarrow{+b} v$), then flow increases by bottleneck capacity b, which is at most $c(e) - f(e)$, so does not exceed $c(e)$
- If e appears in P as a reverse edge ($v \xleftarrow{-b} u$), then flow decreases by bottleneck capacity b, which is at most $f(e)$, so is at least 0

Flow Conservation



- Consider any node v in the augmenting path, and do a case analysis on the types of the incoming and outgoing edge:

residual graph: $P = s \rightsquigarrow u \longrightarrow v \longrightarrow w \rightsquigarrow t$

original graph:

$$u \xrightarrow{+b} v \xrightarrow{+b} w$$
$$u \xrightarrow{+b} v \xleftarrow{-b} w$$
$$u \xleftarrow{-b} v \xrightarrow{+b} w$$
$$u \xleftarrow{-b} v \xleftarrow{-b} w$$

- In all cases, the change in incoming flow to v is equal to the change in outgoing flow from v.

Step 2: Termination and Running Time

Assumption: All capacities are integers. By nature of F-F, all flow values and residual capacities remain integers during the algorithm.

Running time:

- In each F-F iteration, flow increases by at least 1. Therefore, number of iterations is at most $v(f^*)$, where f^* is the final flow.
- Let C be the total capacity of edges leaving source s
- Then $v(f^*) \leq C$.
- So F-F terminates in at most C iterations

Running time per iteration? $O(m + n)$ to find an augmenting path