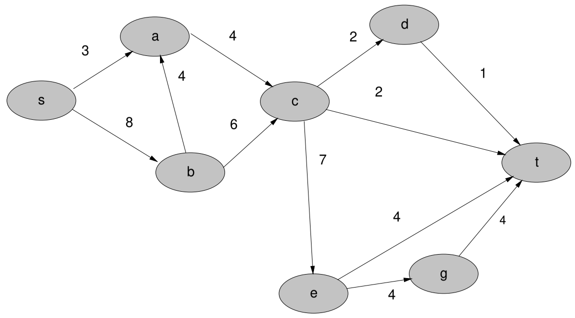
W4231: Analysis of Algorithms

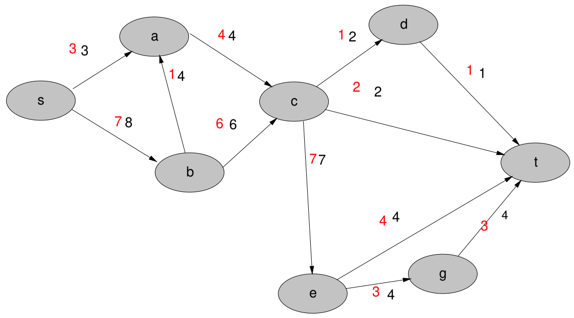
11/3/1999

- Cuts and Flow

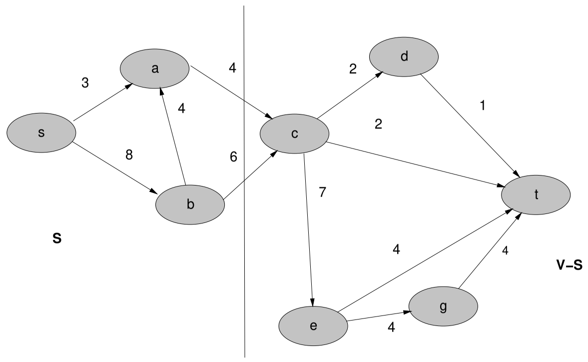
A Network



A Flow in the Network



A Cut in the Network



The Flow Through a Cut is Independent of the Cut

We want to prove:

Theorem 1. *Fix a flow f. For every cut $S, V-S$ we have*

$$\sum_{u\in S, v\notin S, (u,v)\in E} f(u,v) - \sum_{u\in S, v\notin S, (v,u)\in E} f(v,u)$$

is always the same, independently of S.

As a special case, we have that $\sum_u f(s,u) = \sum_v f(v,t)$, so that the cost of a flow is well-defined.

Proof

Assume $f(u,v)$ is defined for every pair (u,v) and $f(u,v)=0$ if $(u,v)\notin E$.

$$\sum_{u\in S, v\notin S} f(u,v) - \sum_{u\notin S, v\in S} f(u,v)$$

$$= \sum_{u\in S, v\in V} f(u,v) - \sum_{u\in S, v\in S} f(u,v) - \sum_{u\notin S, v\in S} f(u,v)$$

$$= \sum_{u\in S, v\in V} f(u,v) - \sum_{u\in V, v\in S} f(u,v)$$

$$= \sum_{v\in V} f(s,v) + \sum_{u\in S-\{s\},v\in V} f(u,v) - \sum_{u\in V,v\in S-\{s\}} f(u,v)$$
$$= \sum_{v\in V} f(s,v)$$

and the last term is independent of S.

Cuts and Flows

If there exists a cut $S, V-S$ of capacity c, then no flow can have cost more than c.

PROOF: consider any flow f. The cost of the flow is

$$\begin{aligned}\sum_v f(s,v) &= \sum_{u\in S, v\notin S} f(u,v) - \sum_{u\notin S, v\in S} f(u,v)\\ &\le \sum_{u\in S,v\notin S} f(u,v)\\ &\le \sum_{u\in S,v\notin S} c_{u,v} = c\end{aligned}$$

Max Flow – Min Cut Theorem

The example with the network where the max flow has cost 10 and there is a cut of cost 10 could have seemed an exceedingly unlikely coincidence. Instead:

Theorem 2. *For every network, there is a cut whose capacity is equal to the cost of the maximum flow.*

Residual Network

Let f be a flow in a network.

The "residual network" with respect to f is the network whose capacities are

$$c^f_{u,v} = \begin{cases} c_{u,v} - f_{u,v} & \text{if } f_{u,v} > 0\\ c_{u,v} + f_{v,u} & \text{if } f_{v,u} > 0\\ c_{u,v} & \text{if } f_{u,v} = f_{v,u} = 0\end{cases}$$

Interpretation

For every pair of vertices u and v, the capacity in the residual network tells us how much further flow we can push from u to v.

If f assigns some flow from v to u, then the residual capacity is bigger than as it was before, since we can "virtually" push flow from v to u by reducing the current flow in the opposite direction.

Proof of the Max Flow/Min Cut Thm

Proof: Fix a maximum flow. Compute the residual network. Call S the set of vertices that are reachable from s using the edges (of non-zero capacity) of the residual network.

- $S, V-S$ is a cut: s belongs to S, and it is impossible that $t \in S$, as otherwise the flow is not maximum.
- the cost of the flow is $\sum_{u\in S,v\notin S} f_{u,v}$ while the capacity of the cut is $\sum_{u\in S,v\notin S} c_{u,v}$.

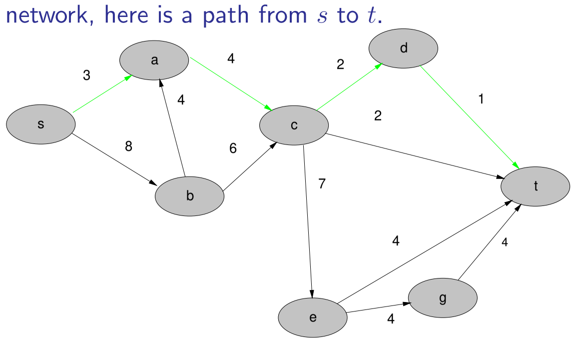
- For every $u \in S$ and $v \notin S$, we must have $f_{u,v} = c_{u,v}$, otherwise v would be reachable from s. We must also have $f_{v,u} = 0$, otherwise also v would be reachable from s.

Ford-Fulkerson Methodology

1. Start from an empty flow.
2. See if t is reachable from s in the residual network.
3. If it is reachable, increase the flow along a path from s to t, recompute residual network, go to 2.
4. If it is not reachable, the proof of the Max Flow/Min Cut theorem shows that the current flow is optimal.

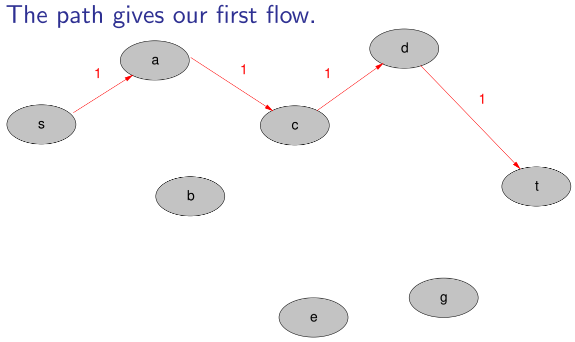
Example — Start

The flow is all-zero, the residual network is equal to the original network, here is a path from s to t.



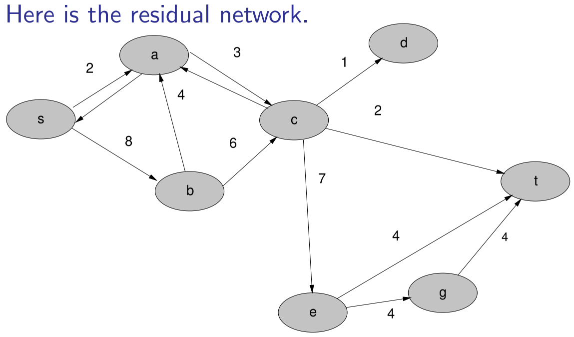
First Flow

The path gives our first flow.

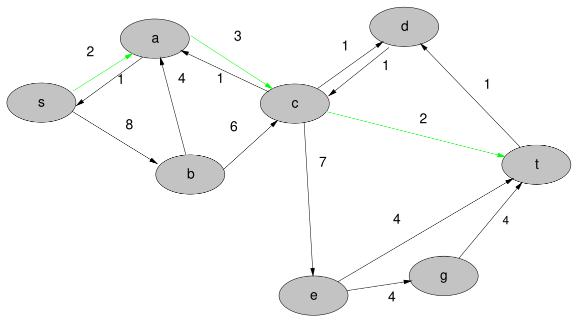


Residual Network

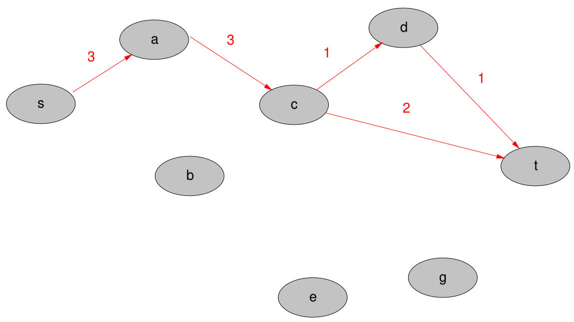
Here is the residual network.



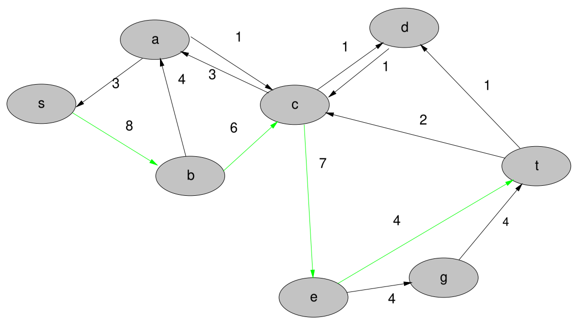
Path in the Residual Network



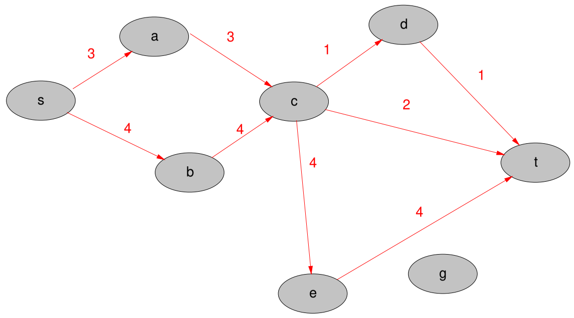
New Flow



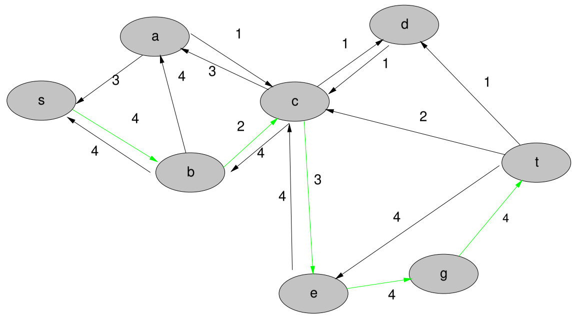
New Residual Network, and a Path in it



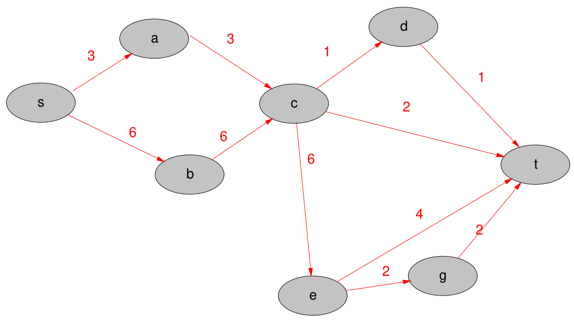
New Flow



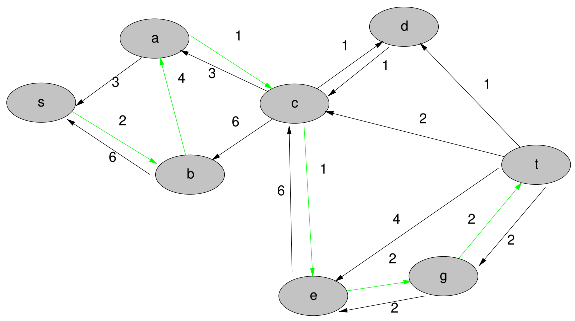
New Residual Network, and a Path in it



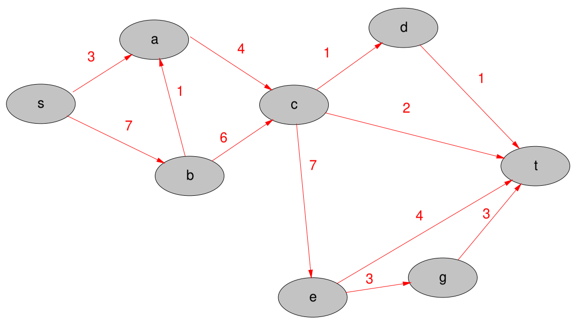
New Flow



New Residual Network

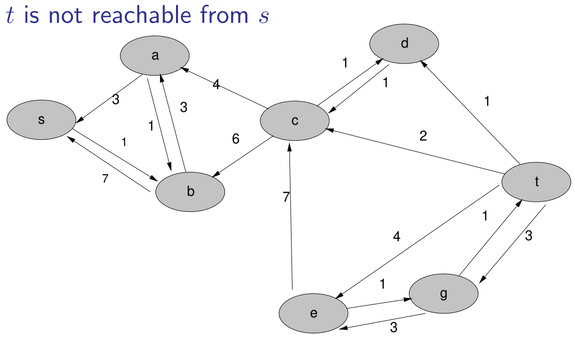


Final Flow

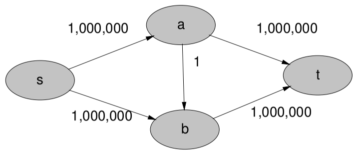


Final Residual Network

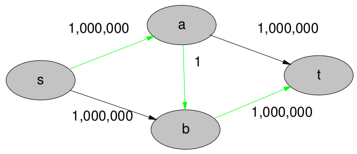
t is not reachable from s



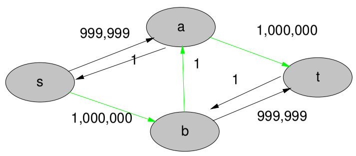
A Bad Case



First Path



Residual Network and Second Path



Edmonds and Karp

Find a path in the residual network **using breadth first search**.

It will be a path with a minimal number of edges.

Theorem 3. *The distance betweem s and t in the residual network never decreases if the Edmonds-Karp algorithm is used.*

Theorem 4. *The Edmonds-Karp algorithm terminates in $O(mn)$ steps.*

Since each phase is just a BFS, that requires $O(m+n) = O(m)$ time, the total time is $O(m^2n)$.

Global Min-Cut

Consider the following problem (called Global Min-Cut):

- Given a (non-weighted) undirected graph $G = (V, E)$, find a non-empty subset of the vertices $S \subseteq V$ such that the number of edges that cross the cut $(S, V - S)$ is minimized.

Difference with Min-Cut as seen before:

1) the vertices s and t are not specified; 2) the graph is not weighted; 3) the graph is undirected.

k-Connectivity

Given an undirected graph G, we say that it is connected if there is a path between any two vertices.

We say that G it is 2-connected if even if we deleted an edge (no matter which one) then G would remain connected.

...

G is k-connected if even if we delete $k - 1$ edges (no matter which ones) then G would remain connected.

Global Min-Cut as Network Fault-Tolerance

A graph is k connected iff its global min cut has cost k.

k-connectedness is an important fault-tolerance property of networks (even if $k - 1$ links fail, we can still route packets from any place to any place).

Using Max Flow

On input G, undirected non-weighted. Make it directed by putting edges (u, v) and (v, u) for every undirected edge $\{u, v\}$. Make it weighted by giving capacity 1 to all the edges.

If we knew two vertices s and t such that $s \in S$ and $t \notin S$ in a global min-cut S, then we could find S with a max-flow computation.

We can try all the pair of vertices s and t, and find the max-flow (and thus the min-cut) each time.

In fact we can just fix s arbitrarily and try all t. This is still n max-flow computations.

A Randomized Algorithm

We will present a randomized algorithm for global min-cut that runs in time $O(n^2)$ and has a probability $1/n^2$ of finding the global min-cut.

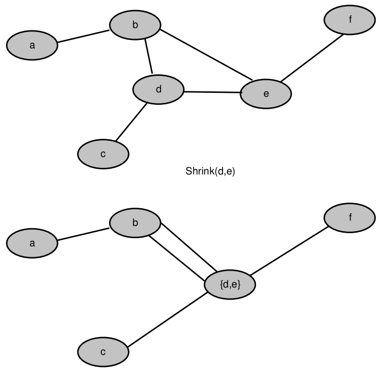
Then it can be modified to run in time $O(n^2(\log n)^{O(1)})$ and be correct with very high probability.

The Shrink operator

The algorithm consists in repeated application of the $Shrink(u, v)$ procedure, where (u, v) is an edge of the graph.

$Shrink(u, v)$ transforms a graph into a new one where in place of the vertices u and v there is a new vertex $\{u, v\}$, and the new vertex is adjacent to all the vertices that were previously adjacent to u or v. If u and v had a common neighbor z, there will be two "parallel" edges connecting $\{u, v\}$ to z.

Example



Algorithm

- While there are more than two vertices:
  - Choose at random one edge (u, v), and do $Shrink(u, v)$.
- The cut is given by the set of original vertices that have collapsed into one of the two final vertices.

Running Time

The number of vertices in the graph decreases by 1 at each iteration. There are $n - 2$ iterations.

Each iteration can be implemented in $O(n)$ time.

Correctness

Let $S, V - S$ be a global minimum cut. If the algorithm never chooses an edge that crosses the cut, then the algorithm is correct. (The algorithm will collapse all the elements of S into one of the final macro-vertices, and $V - S$ in the other macro-vertices.)

We have to show that there is a probability at least $1/n^2$ that this happens.

Let k be the number of edges in the global min-cut.

Then every vertex in G has degree at least k.

Then G has at least $nk/2$ edges.

Then the probability that we choose one of those that are not in the minimum cut is at least $(1 - 2/n)$.

After the first $Shrink()$ we are left with $n - 1$ vertices. Every "macro-vertex" must have degree at least k. So the number of edges is at least $(n - 1)k/2$, and the probability that we choose one of those that are not in the minimum cut is at least $(1 - 2/(n - 1))$.

. . .

The probability that everything goes well is at least

$$\left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\cdots\left(1 - \frac{2}{3}\right)$$

Which is

$$\frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdots \frac{1}{3} = 1/\binom{n}{2} > \frac{2}{n^2}$$