Network Flow Algorithms

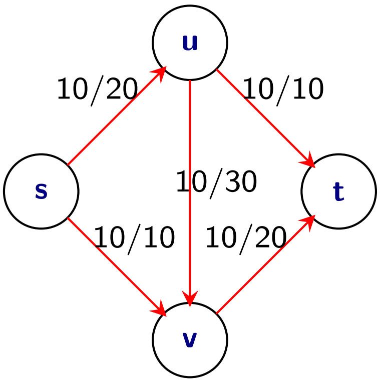
Lecture 17

March 29, 2011

Part I

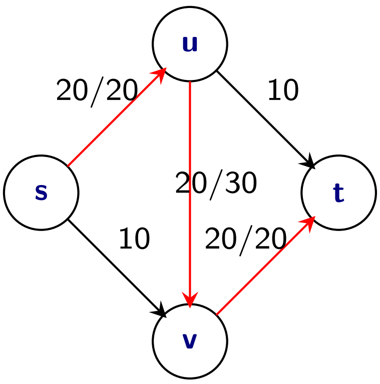
Algorithm(s) for Maximum Flow

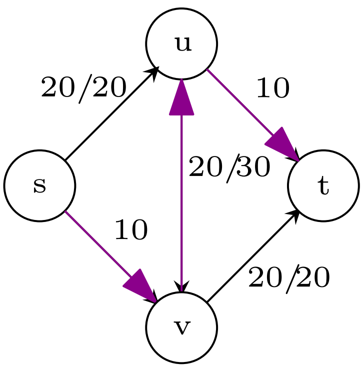
Greedy Approach



1. Begin with $f(e) = 0$ for each edge
2. Find a s-t path P with $f(e) < c(e)$ for every edge $e \in P$
3. *Augment* flow along this path
4. Repeat augmentation for as long as possible.

Greedy Approach: Issues





1. Begin with $f(e) = 0$ for each edge
2. Find a s-t path P with $f(e) < c(e)$ for every edge $e \in P$
3. Augment flow along this path
4. Repeat augmentation for as long as possible.

Greedy can get stuck in sub-optimal flow!

Need to "push-back" flow along edge (u, v)

Residual Graph

Definition

For a network $\mathbf{G = (V, E)}$ and flow $\mathbf{f}$, the residual graph $\mathbf{G_f = (V', E')}$ of $\mathbf{G}$ with respect to $\mathbf{f}$ is

- $\mathbf{V' = V}$
- Forward Edges: For each edge $\mathbf{e \in E}$ with $\mathbf{f(e) < c(e)}$, we $\mathbf{e \in E'}$ with capacity $\mathbf{c(e) - f(e)}$
- Backward Edges: For each edge $\mathbf{e = (u, v) \in E}$ with $\mathbf{f(e) > 0}$, we $\mathbf{(v, u) \in E'}$ with capacity $\mathbf{f(e)}$

Residual Graph Example

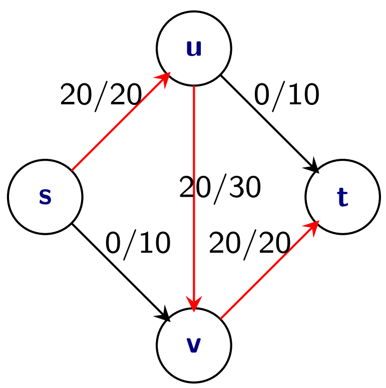


Figure: Flow in red edges

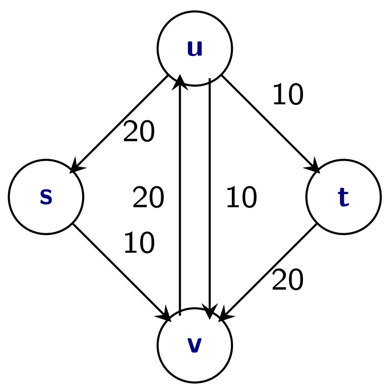


Figure: Residual Graph

Residual Graph Property

Observation: Residual graph captures the "residual" problem exactly.

Lemma

Let $\mathbf{f}$ be a flow in $\mathbf{G}$ and $\mathbf{G_f}$ be the residual graph. If $\mathbf{f'}$ is a flow in $\mathbf{G_f}$ then $\mathbf{f} + \mathbf{f'}$ is a flow in $\mathbf{G}$ of value $\mathbf{v(f)} + \mathbf{v(f')}$.

Lemma

Let $\mathbf{f}$ and $\mathbf{f'}$ be two flows in $\mathbf{G}$ with $\mathbf{v(f')} \geq \mathbf{v(f)}$. Then there is a flow $\mathbf{f''}$ of value $\mathbf{v(f')} - \mathbf{v(f)}$ in $\mathbf{G_f}$.

Definition of $+$ and - for flows is intuitive and the above lemmas are easy in some sense but a bit messy to formally prove.

Residual Graph Property: Implication

Recursive algorithm for finding a maximum flow:

```
MaxFlow(G, s, t):
    If the flow from s to t is 0
        return 0
    Find any flow f with v(f) > 0 in G
    Recursively compute a maximum flow f' in G_f
    Output the flow f + f'
```

Iterative algorithm for finding a maximum flow:

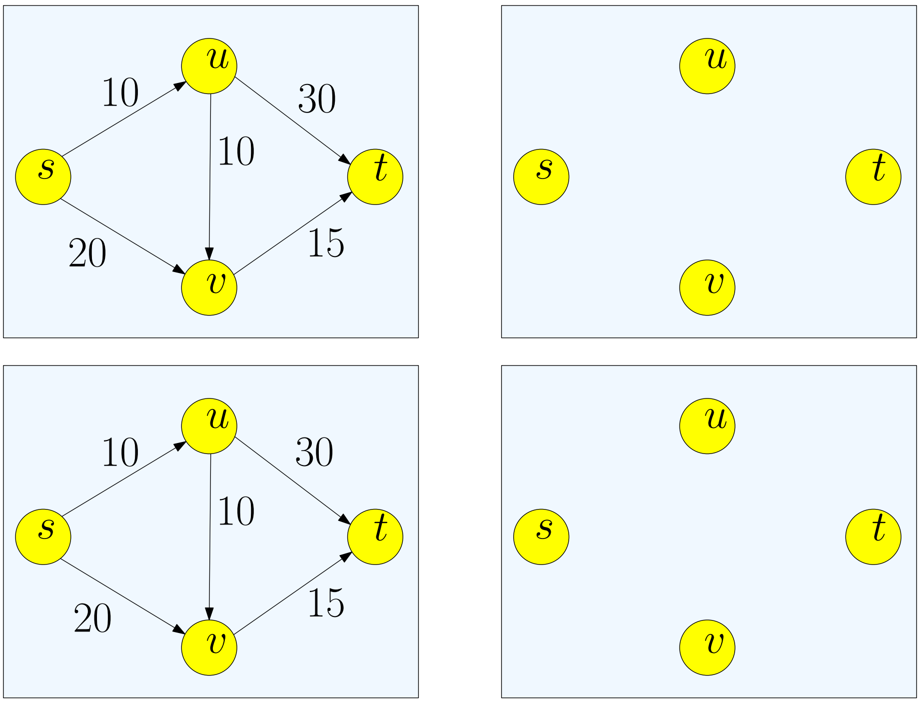
```
MaxFlow(G, s, t):
    Start with flow f that is 0 on all edges
    While there is a flow f' in G_f with v(f') > 0 do
        f = f + f'
        Update G_f
    endWhile
    Output f
```

Ford-Fulkerson Algorithm

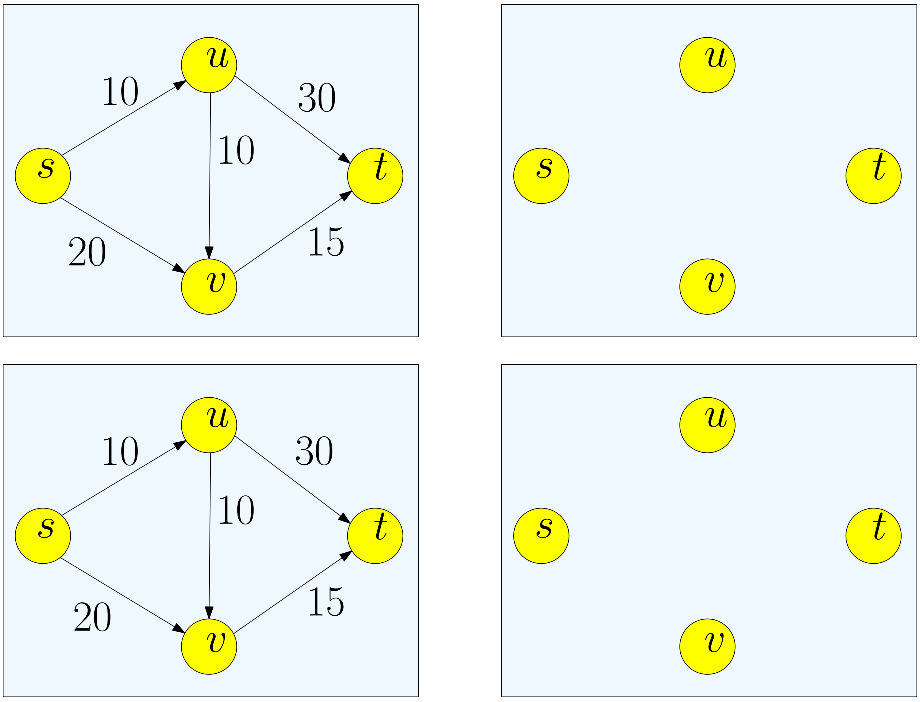
algFordFulkerson
    for every edge $\mathbf{e}$, $\mathbf{f(e) = 0}$
    $\mathbf{G_f}$ is residual graph of $\mathbf{G}$ with respect to $\mathbf{f}$
    while $\mathbf{G_f}$ has a simple $\mathbf{s}$-$\mathbf{t}$ path **do**
        let $\mathbf{P}$ be simple $\mathbf{s}$-$\mathbf{t}$ path in $\mathbf{G_f}$
        $\mathbf{f = augment(f, P)}$
        Construct new residual graph $\mathbf{G_f}$

augment$(\mathbf{f}, \mathbf{P})$
    let $\mathbf{b}$ be bottleneck capacity,
        i.e., min capacity of edges in $\mathbf{P}$ (in $\mathbf{G_f}$)
    for each edge $\mathbf{(u, v)}$ in $\mathbf{P}$ **do**
        if $\mathbf{e = (u, v)}$ is a forward edge **then**
            $\mathbf{f(e) = f(e) + b}$
        else (* $\mathbf{(u, v)}$ is a backward edge *)
            let $\mathbf{e = (v, u)}$ (* $\mathbf{(v, u)}$ is in $\mathbf{G}$ *)
            $\mathbf{f(e) = f(e) - b}$
    return $\mathbf{f}$

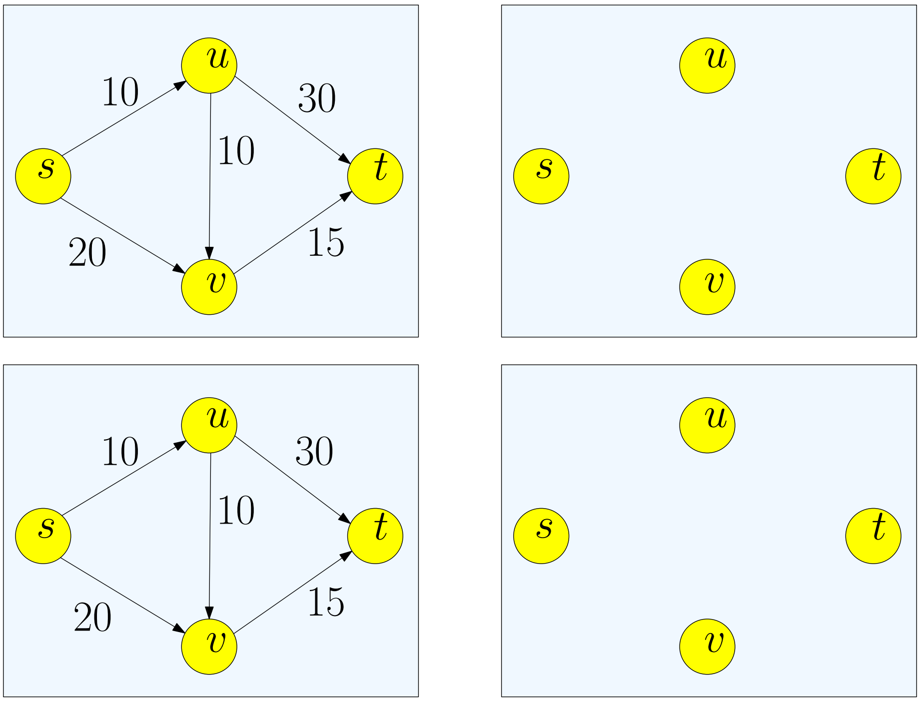
Example



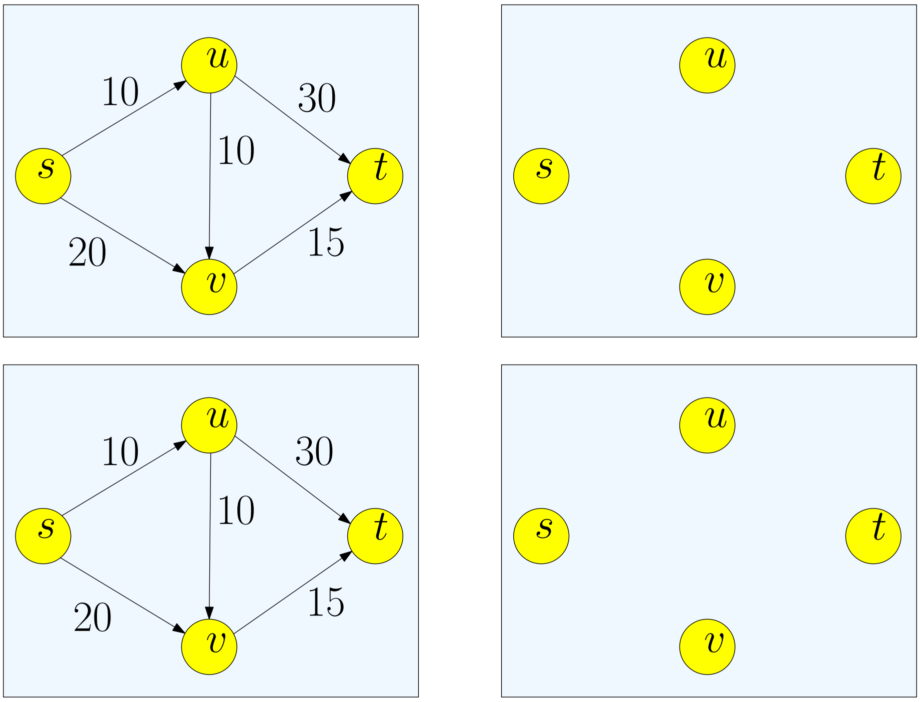
Example continued



Example continued



Example continued



Properties about Augmentation: Flow

Lemma

If $\mathbf{f}$ *is a flow and* $\mathbf{P}$ *is a simple* $\mathbf{s}$-$\mathbf{t}$ *path in* $\mathbf{G_f}$, *then* $\mathbf{f'} = \mathtt{augment}(\mathbf{f}, \mathbf{P})$ *is also a flow.*

Proof.

Verify that $\mathbf{f'}$ is a flow. Let $\mathbf{b}$ be augmentation amount.

- Capacity constraint: If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a forward edge then $\mathbf{f'(e)} = \mathbf{f(e)} + \mathbf{b}$ and $\mathbf{b} \leq \mathbf{c(e)} - \mathbf{f(e)}$. If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a backward edge, then letting $\mathbf{e} = (\mathbf{v}, \mathbf{u})$, $\mathbf{f'(e)} = \mathbf{f(e)} - \mathbf{b}$ and $\mathbf{b} \leq \mathbf{f(e)}$. Both cases $\mathbf{0} \leq \mathbf{f'(e)} \leq \mathbf{c(e)}$.
- Conservation constraint: Let $\mathbf{v}$ be an internal node. Let $\mathbf{e_1}, \mathbf{e_2}$ be edges of $\mathbf{P}$ incident to $\mathbf{v}$. Four cases based on whether $\mathbf{e_1}, \mathbf{e_2}$ are forward or backward edges. Check cases (see fig next slide). $\square$

Properties about Augmentation: Conservation Constraint

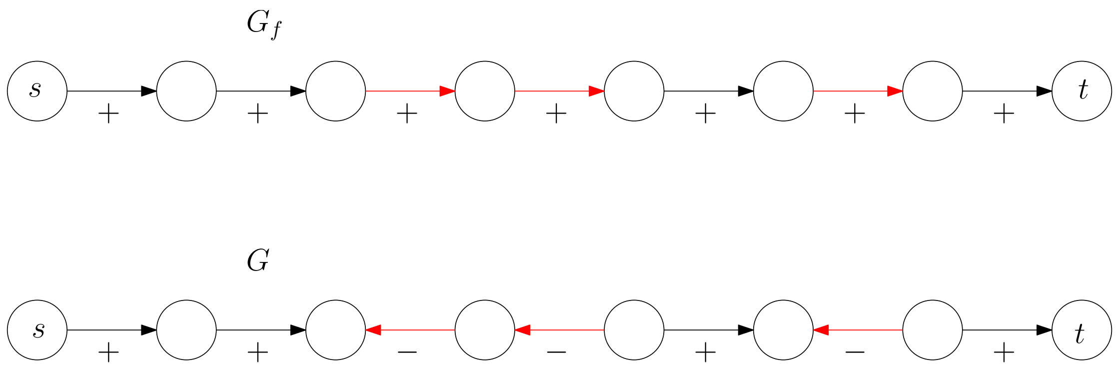


Figure: Augmenting path $\mathbf{P}$ in $\mathbf{G_f}$ and corresponding change of flow in $\mathbf{G}$. Red edges are backward edges.

Properties about Augmentation: Integer Flow

Lemma

At every stage of the Ford-Fulkerson algorithm, the flow values $\mathbf{f(e)}$ *and the residual capacities in* $\mathbf{G_f}$ *are integers*

Proof.

Initial flow and residual capacities are integers. Suppose lemma holds for $\mathbf{j}$ iterations. Then in $\mathbf{(j+1)}$st iteration, minimum capacity edge $\mathbf{b}$ is an integer, and so flow after augmentation is an integer. $\square$

Progress in Ford-Fulkerson

Proposition

Let $\mathbf{f}$ *be a flow and* $\mathbf{f'}$ *be flow after one augmentation. Then* $\mathbf{v(f) < v(f')}$.

Proof.

Let $\mathbf{P}$ be an augmenting path, i.e., $\mathbf{P}$ is a simple $\mathbf{s}$-$\mathbf{t}$ path in residual graph

- First edge $\mathbf{e}$ in $\mathbf{P}$ must leave $\mathbf{s}$
- Original network $\mathbf{G}$ has no incoming edges to $\mathbf{s}$; hence $\mathbf{e}$ is a forward edge
- $\mathbf{P}$ is simple and so never returns to $\mathbf{s}$
- Thus, value of flow increases by the flow on edge $\mathbf{e}$ □

Termination Proof

Theorem

Let $\mathbf{C}$ *be the minimum cut value; in particular* $\mathbf{C} \leq \sum_{\mathbf{e} \text{ out of } \mathbf{s}} \mathbf{c(e)}$. *Ford-Fulkerson algorithm terminates after finding at most* $\mathbf{C}$ *augmenting paths.*

Proof.

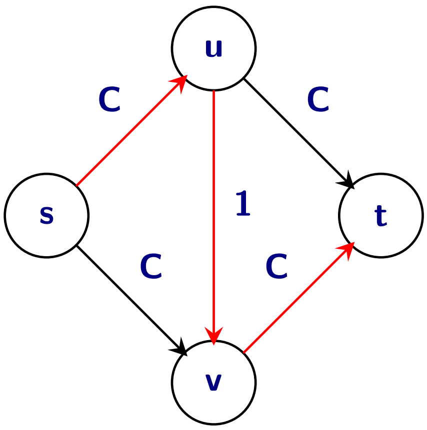
The value of the flow increases by at least $\mathbf{1}$ after each augmentation. Maximum value of flow is at most $\mathbf{C}$. □

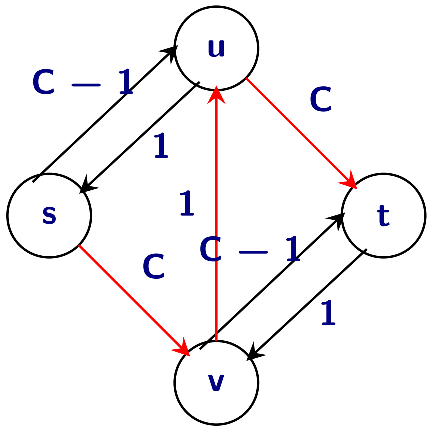
Running time

- Number of iterations $\leq \mathbf{C}$
- Number of edges in $\mathbf{G_f} \leq \mathbf{2m}$
- Time to find augmenting path is $\mathbf{O(n + m)}$
- Running time is $\mathbf{O(C(n + m))}$ (or $\mathbf{O(mC)}$).

Efficiency of Ford-Fulkerson

Running time $= O(mC)$ is not polynomial. Can the running time be as $\Omega(mC)$ or is our analysis weak?





Ford-Fulkerson can take $\Omega(C)$ iterations.

Correctness of Ford-Fulkerson Augmenting Path Algorithm

Question: When the algorithm terminates, is the flow computed the maximum s-t flow?

Proof idea: show a cut of value equal to the flow. Also shows that maximum flow is equal to minimum cut!

Recalling Cuts

Definition

Given a flow network an $\mathbf{s}$-$\mathbf{t}$ cut is a set of edges $\mathbf{E}' \subset \mathbf{E}$ such that removing $\mathbf{E}'$ *disconnects* $\mathbf{s}$ from $\mathbf{t}$: in other words there is no directed $\mathbf{s} \to \mathbf{t}$ path in $\mathbf{E} - \mathbf{E}'$. *Capacity* of cut $\mathbf{E}'$ is $\sum_{\mathbf{e} \in \mathbf{E}'} \mathbf{c}(\mathbf{e})$.

Let $\mathbf{A} \subset \mathbf{V}$ such that

- $\mathbf{s} \in \mathbf{A}$, $\mathbf{t} \not\in \mathbf{A}$
- $\mathbf{B} = \mathbf{V} - \mathbf{A}$ and hence $\mathbf{t} \in \mathbf{B}$

Define $(\mathbf{A}, \mathbf{B}) = \{(\mathbf{u}, \mathbf{v}) \in \mathbf{E} \mid \mathbf{u} \in \mathbf{A}, \mathbf{v} \in \mathbf{B}\}$

Claim

$(\mathbf{A}, \mathbf{B})$ *is an* $\mathbf{s}$-$\mathbf{t}$ *cut.*

Recall: Every *minimal* $\mathbf{s}$-$\mathbf{t}$ cut $\mathbf{E}'$ is a cut of the form $(\mathbf{A}, \mathbf{B})$.

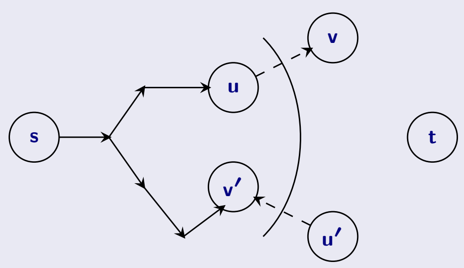
Ford-Fulkerson Correctness

Lemma

If there is no $\mathbf{s}$-$\mathbf{t}$ *path in* $\mathbf{G_f}$ *then there is some cut* $(\mathbf{A}, \mathbf{B})$ *such that* $\mathbf{v}(\mathbf{f}) = \mathbf{c}(\mathbf{A}, \mathbf{B})$

Proof.

Let $\mathbf{A}$ be all vertices reachable from $\mathbf{s}$ in $\mathbf{G_f}$; $\mathbf{B} = \mathbf{V} \setminus \mathbf{A}$

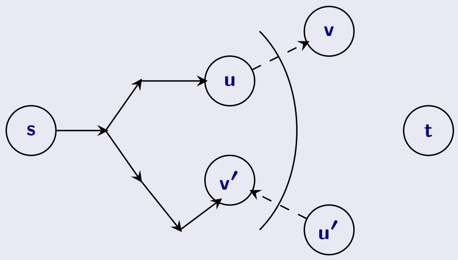


- $\mathbf{s} \in \mathbf{A}$ and $\mathbf{t} \in \mathbf{B}$. So $(\mathbf{A}, \mathbf{B})$ is an $\mathbf{s}$-$\mathbf{t}$ cut in $\mathbf{G}$
- If $\mathbf{e} = (\mathbf{u}, \mathbf{v}) \in \mathbf{G}$ with $\mathbf{u} \in \mathbf{A}$ and $\mathbf{v} \in \mathbf{B}$, then $\mathbf{f}(\mathbf{e}) = \mathbf{c}(\mathbf{e})$ (saturated edge) because otherwise $\mathbf{v}$ is reachable from $\mathbf{s}$ in $\mathbf{G_f}$

□

Lemma Proof Continued

Proof.

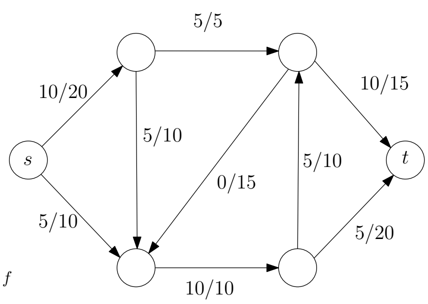


- If $\mathbf{e = (u', v') \in G}$ with $\mathbf{u' \in B}$ and $\mathbf{v' \in A}$, then $\mathbf{f(e) = 0}$ because otherwise $\mathbf{u'}$ is reachable from $\mathbf{s}$ in $\mathbf{G_f}$
- Thus,

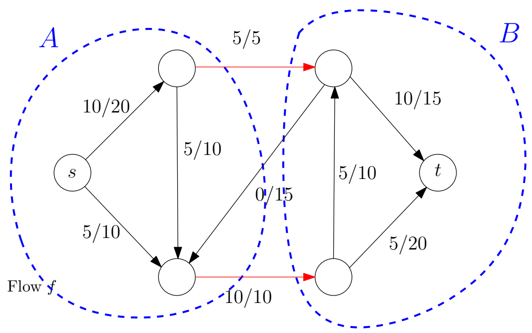
$$\begin{aligned} \mathbf{v(f)} &= \mathbf{f^{out}(A) - f^{in}(A)} \\ &= \mathbf{f^{out}(A) - 0} \\ &= \mathbf{c(A, B) - 0} \\ &= \mathbf{c(A, B)} \end{aligned}$$

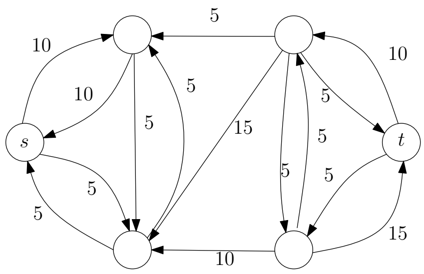
□

Example

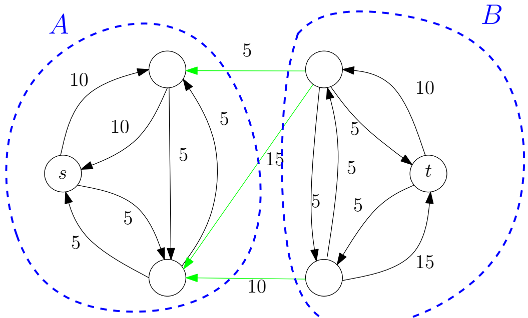


Flow f





Residual graph G_f: no s-t path



A is reachable set from s in G_f

Ford-Fulkerson Correctness

Theorem

The flow returned by the algorithm is the maximum flow.

Proof.

- For any flow $\mathbf{f}$ and $\mathbf{s}$-$\mathbf{t}$ cut $(\mathbf{A}, \mathbf{B})$, $\mathbf{v}(\mathbf{f}) \leq \mathbf{c}(\mathbf{A}, \mathbf{B})$
- For flow $\mathbf{f}^*$ returned by algorithm, $\mathbf{v}(\mathbf{f}^*) = \mathbf{c}(\mathbf{A}^*, \mathbf{B}^*)$ for some $\mathbf{s}$-$\mathbf{T}$ cut $(\mathbf{A}^*, \mathbf{B}^*)$
- Hence, $\mathbf{f}^*$ is maximum

□

Max-Flow Min-Cut Theorem and Integrality of Flows

Theorem

For any network $\mathbf{G}$, the value of a maximum $\mathbf{s}$-$\mathbf{t}$ flow is equal to the capacity of the minimum $\mathbf{s}$-$\mathbf{t}$ cut.

Proof.

Ford-Fulkerson algorithm terminates with a maximum flow of value equal to the capacity of a (minimum) cut. □

Max-Flow Min-Cut Theorem and Integrality of Flows

Theorem

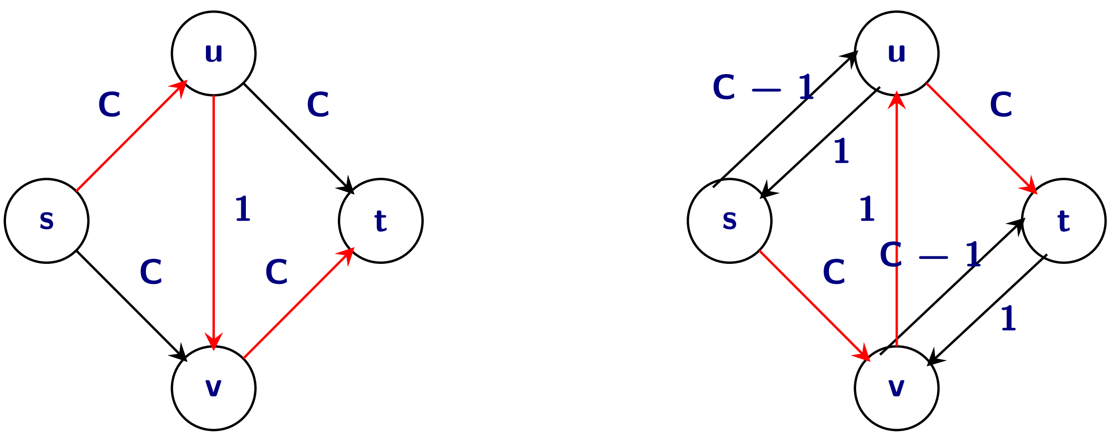
For any network $\mathbf{G}$ *with integer capacities, there is a maximum* $\mathbf{s}$-$\mathbf{t}$ *flow that is integer valued.*

Proof.

Ford-Fulkerson algorithm produces an integer valued flow when capacities are integers. □

Efficiency of Ford-Fulkerson

Running time $= \mathbf{O(mC)}$ is not polynomial. Can the upper bound be achieved?



Polynomial Time Algorithms

Question: Is there a polynomial time algorithm for maxflow?

Question: Is there a variant of Ford-Fulkerson that leads to a polynomial time algorithm? Can we choose an augmenting path in some clever way? Yes! Two variants.

- Choose the augmenting path with largest bottleneck capacity.
- Choose the shortest augmenting path.

Augmenting Paths with Large Bottleneck Capacity

- Pick augmenting paths with largest bottleneck capacity in each iteration of Ford-Fulkerson
- How do we find path with largest bottleneck capacity?
  - Assume we know Δ the bottleneck capacity
  - Remove all edges with residual capacity $\leq \Delta$
  - Check if there is a path from s to t
  - Do binary search to find largest Δ
  - Running time: $O(m \log C)$
- Can we bound the number of augmentations? Can show that in $O(m \log C)$ augmentations the algorithm reaches a max flow. This leads to an $O(m^2 \log^2 C)$ time algorithm.

Augmenting Paths with Large Bottleneck Capacity

How do we find path with largest bottleneck capacity?

- Max bottleneck capacity is one of the edge capacities. Why?
- Can do binary search on the edge capacities. First, sort the edges by their capacities and then do binary search on that array as before.
- Algorithm's running time is $O(m \log m)$.
- Different algorithm that also leads to $O(m \log m)$ time algorithm by adapting Prim's algorithm.

Removing Dependence on C

- [Edmonds-Karp, Dinitz] Picking augmenting paths with fewest number of edges yields a $O(m^2 n)$ algorithm, i.e., independent of C. Such an algorithm is called a strongly polynomial time algorithm since the running time does not depend on the numbers (assuming RAM model). (Many implementation of Ford-Fulkerson would actually use shortest augmenting path if they use BFS to find an s-t path).
- Further improvements can yield algorithms running in $O(mn \log n)$, or $O(n^3)$.

Finding a Minimum Cut

Question: How do we find an actual minimum s-t cut?
Proof gives the algorithm!

- Compute an s-t maximum flow f in G
- Obtain the residual graph G_f
- Find the nodes A reachable from s in G_f
- Output the cut $(A, B) = \{(u, v) \mid u \in A, v \in B\}$. Note: The cut is found in G while A is found in G_f

Running time is essentially the same as finding a maximum flow.

Note: Given G and a flow f there is a linear time algorithm to check if f is a maximum flow and if it is, outputs a minimum cut. How?