Common Features of Flow Networks

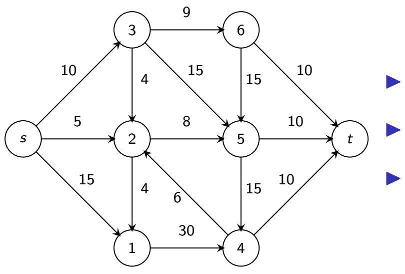
- *Network* represented by a (directed) *graph* $G = (V, E)$
- Each edge e has a *capacity* $c(e) \geq 0$ that limits amount of *traffic* on e
- *Source(s)* of traffic/data
- *Sink(s)* of traffic/data
- Traffic *flows* from sources to sinks
- Traffic is *switched/interchanged* at nodes

Flow: abstract term to indicate stuff (traffic/data/etc) that *flows* from sources to sinks.

Single Source Single Sink Flows

Simple setting:

- single source s and single sink t
- every other node v is an *internal* node
- flow originates at s and terminates at t



- Each edge e has a capacity $c(e) \geq 0$
- Source $s \in V$ with no incoming edges
- Sink $t \in V$ with no outgoing edges

Assumptions: All capacities are integer, and every vertex has at least one edge incident to it.

Definition of Flow

Two ways to define flows:

- edge based
- path based

They are essentially equivalent but have different uses.

Edge based definition is more compact.

Edge Based Definition of Flow

Definition

A flow in a network $G = (V, E)$, is a function $f : E \to \mathbb{R}^{\geq 0}$ such that

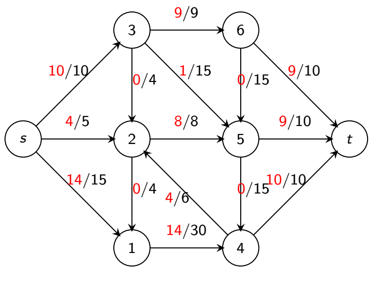


Figure : Flow with value

- Capacity Constraint: For each edge e, $f(e) \leq c(e)$
- Conservation Constraint: For each vertex $v \neq s, t$

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

More Definitions and Notation

Notation

- The inflow into a vertex v is $f^{\text{in}}(v) = \sum_{e \text{ into } v} f(e)$ and the outflow is $f^{\text{out}}(v) = \sum_{e \text{ out of } v} f(e)$
- For a set of vertices A, $f^{\text{in}}(A) = \sum_{e \text{ into } A} f(e)$. Outflow $f^{\text{out}}(A)$ is defined analogously

Definition

For a network $G = (V, E)$ with source s, the value of flow f is defined as $v(f) = f^{\text{out}}(s)$

A Path Based Definition of Flow

Intuition: flow goes from source s to sink t along a path.

$\mathcal{P}$: set of all paths from s to t. $|\mathcal{P}|$ can be *exponential* in n!

Definition

A flow in a network $G = (V, E)$, is a function $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ such that

- Capacity Constraint: For each edge e, total flow on e is $\leq c(e)$.

$$\sum_{p \in \mathcal{P}_e} f(p) \leq c(e)$$

- Conservation Constraint: No need! Automatic.

Value of flow: $\sum_{p \in \mathcal{P}} f(p)$

Path based flow implies Edge based flow

Lemma

Given a path based flow $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ there is an edge based flow $f' : E \to \mathbb{R}^{\geq 0}$ of the same value.

Proof.

For each edge e define $f'(e) = \sum_{p : e \in p} f(p)$.
Verify capacity and conservation constraints for f'. □

Edge based flow to Path based Flow

Flow Decomposition:

Lemma

Given an edge based flow $f' : E \to \mathbb{R}^{\geq 0}$, there is a path based flow $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ of same value. Moreover, f assigns non-negative flow to at most $m + n$ paths where $|E| = m$ and $|V| = n$. Given f', the path based flow can be computed in $O(mn)$ time.

Proof Idea.

- remove all edges with $f'(e) = 0$
- find a path p from s to t
- assign $f(p)$ to be $\min_{e \in p} f'(e)$
- reduce $f'(e)$ for all $e \in p$ by $f'(e)$
- repeat until no path from s to t

□

Edge vs Path based Definitions of Flow

Edge based flows:

- *compact* representation, only m values to be specified
- need to check flow conservation explicitly at each internal node

Path flows:

- in some applications, paths more natural
- not compact
- no need to check flow conservation constraints

Equivalence shows that we can go back and forth easily.

The Maximum-Flow Problem

Problem

Input A network G with capacity c and source s and sink t

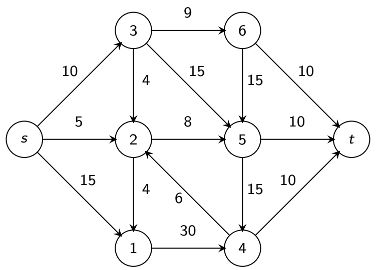
Goal Find flow of *maximum* value

Question: Given a flow network, what is an *upper bound* on the maximum flow between source and sink?

Cuts

Definition

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

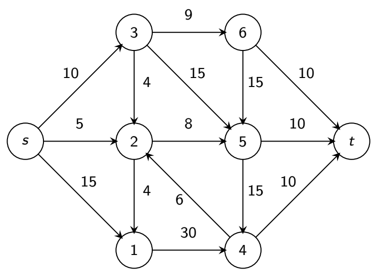


Caution: cut may leave $t \to s$ paths!

Minimum Cut

Definition

Given a flow network an $s-t$ *minimum* cut is a cut E' of smallest capacity amongst all $s-t$ cuts.



Observation: exponential number of $s-t$ cuts and no “easy” algorithm to find a minimum cut.

The Minimum-Cut Problem

Problem

Input: A network G with capacity c and source s and sink t

Goal: Find the capacity of a *minimum* $s - t$ cut

Flows and Cuts

Lemma

For any $s - t$ *cut* E', maximum $s - t$ *flow* $\leq$ *capacity of* E'.

Proof.

Formal proof easier with path based definition of flow.
Suppose $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ is a max-flow.

Every path $p \in \mathcal{P}$ contains an edge $e \in E'$. Why?
Assign each path $p \in \mathcal{P}$ to exactly on edge $e \in E'$.
Let $\mathcal{P}_e$ be paths assigned to $e \in E'$. Then

$$\begin{aligned} v(f) = \sum_{p \in \mathcal{P}} f(p) &= \sum_{e \in E'} \sum_{p \in \mathcal{P}_e} f(p) \\ &\leq \sum_{e \in E'} c(e) \end{aligned}$$

□

Flows and Cuts

Lemma

For any $s - t$ *cut* E', maximum $s - t$ *flow* $\leq$ *capacity of* E'.

Corollary

Maximum $s - t$ *flow* $\leq$ *minimum* $s - t$ *cut.*

Max-Flow Min-Cut Theorem

Theorem

In any flow network the maximum $s - t$ *flow is* equal *to the minimum* $s - t$ *cut.*

Can compute minimum-cut from maximum flow and vice-versa!
Proof coming shortly.

Many applications:

- optimization
- graph theory
- combinatorics

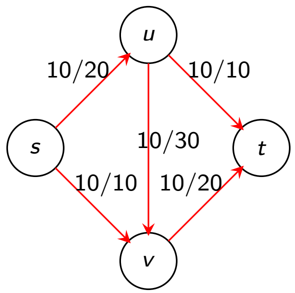
The Maximum-Flow Problem

Problem

Input A network G with capacity c and source s and sink t

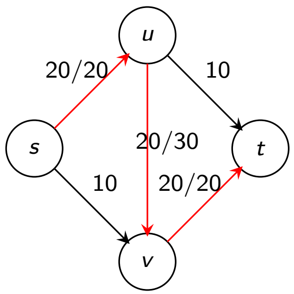
Goal Find flow of *maximum* value

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.

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

Residual Graph

Definition

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

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

Residual Graph Example

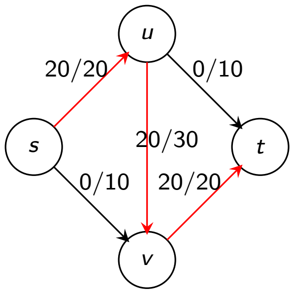


Figure : Flow in red edges

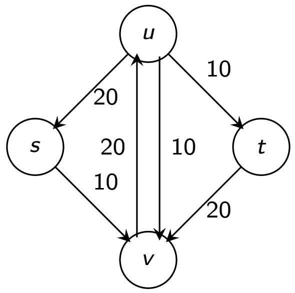


Figure : Residual Graph

Ford-Fulkerson Algorithm

```
for every edge e, f(e) = 0
G_f is residual graph of G with respect to f
while G_f has a simple s-t path
    let P be simple s-t path in G_f
    f = augment(f,P)
    Construct new residual graph G_f

augment(f,P)
    let b be bottleneck capacity, i.e., min capacity of edges in P
    for each edge e in P
        if e is a forward edge
            f(e) = f(e) + b
        else (* e is a backward edge *)
            f(e) = f(e) - b
    return f
```

Properties about Augmentation

Lemma

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

Lemma

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

Proof.

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

Progress in Ford-Fulkerson

Proposition

Let f be a flow and f' be flow after one augmentation. Then $v(f) < v(f')$

Proof.

Let P be an augmenting path, i.e., P is a simple s-t path in residual graph

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

Termination Proof

Theorem
Let $C = \sum_{e \text{ out of } s} c(e)$. Ford-Fulkerson algorithm terminates after finding at most C augmenting paths

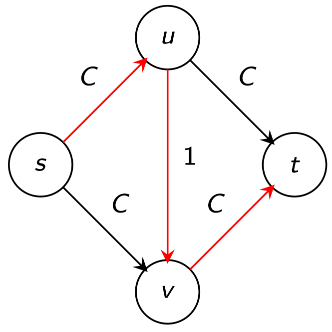
Proof.
The value of the flow increases by at least 1 after each augmentation. Maximum value of flow is at most C. □

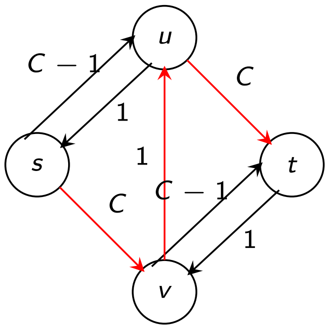
Running time

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

Efficiency of Ford-Fulkerson

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





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!

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

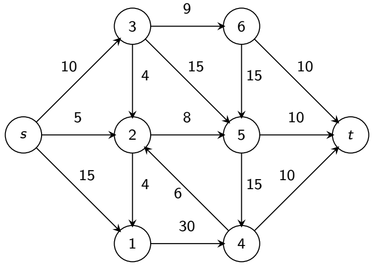
Cuts as Vertex Partitions

Let $A \subset V$ such that

- $s \in A$, $t \notin A$
- $B = V - A$ and hence $t \in B$

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

Claim
(A, B) is an $s - t$ cut.



Cuts as Vertex Partitions

Lemma

Suppose E' is an $s - t$ cut. Then there is a cut (A, B) such that $(A, B) \subseteq E'$.

Proof.

E' is an $s - t$ cut implies no path from s to t in $(V, E - E')$.
Let A be set of all nodes reachable by s in $(V, E - E')$. By above, $t \notin A$.
And also $(A, B) \subseteq E'$. Why? □

Corollary

Every minimal *$s - t$ cut E' is a cut of the form (A, B).*

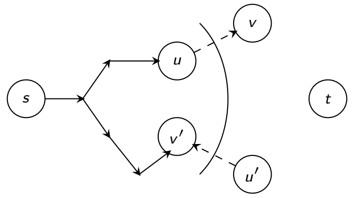
Ford-Fulkerson Correctness

Lemma

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

Proof.

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

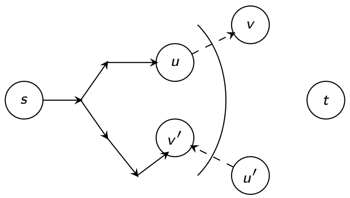


- $s \in A$ and $t \in B$. So (A, B) is an s-t cut in G
- If $e = (u, v) \in G$ with $u \in A$ and $v \in B$, then $f(e) = c(e)$ because otherwise v is reachable from s

□

Lemma Proof Continued

Proof.



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

□

Ford-Fulkerson Correctness

Theorem

The flow returned by the algorithm is the maximum flow.

Proof.

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

□

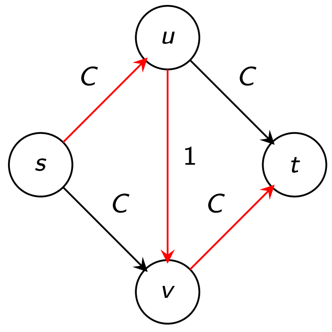
Max-Flow Min-Cut Theorem and Integrality of Flows

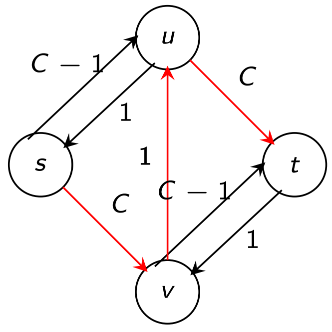
Theorem
For any network G, the value of a maximum $s-t$ flow is equal to the capacity of the minimum s-t cut.

Theorem
For any network G with integer capacities, there is a maximum $s-t$ flow that is integer valued.

Efficiency of Ford-Fulkerson

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





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: $O(m \log C)$

Algorithm works in polynomial time but can devise a simpler algorithm.

Definition
Given graph G, $s-t$ flow f and a parameter Δ, the graph $G_f(\Delta)$ is the residual graph with all edges in G_f with residual capacity $< \Delta$ removed.

Capacity Scaling Algorithm

```
for all edges e, f(e) = 0
Δ = largest power of 2 smaller than maximum capacity edge in G
while Δ ≥ 1
    while there is a simple s-t path in G_f(Δ)
        let P be simple s-t path in G_f(Δ)
        f = augment(f,P)
        update G_f(Δ)
    Δ = Δ/2
```

- Flows and residual capacities are always integral
- When $\Delta = 1$, $G_f(\Delta) = G_f$; so on termination f is max-flow
- Outermost loop runs for at most $\lceil \log C \rceil + 1$
- Each augmentation increases flow by at least Δ

Running Time Analysis of Capacity Scaling Algorithm

- In each scaling phase there are at most $2m$ augmentations
- Each augmenting path can be found in $O(m)$ time
- There are at most $\lceil \log C \rceil + 1$ scaling phases
- Total time is $O(m^2 \log C)$

Paths in $G_f(\Delta)$ and max-flows

Lemma

Let f be such that $G_f(\Delta)$ does not have an s-t augmenting path. Then maximum flow is at most $v(f) + m\Delta$.

Proof.

We will show that there is a cut (A, B) of capacity at most $v(f) + m\Delta$

- Let A be all vertices reachable from s in $G_f(\Delta)$, and let $B = V \setminus A$; (A, B) is an s-t cut

$$\begin{aligned} v(f) &= \textstyle\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) \\ &\geq \textstyle\sum_{e \text{ out of } A} [c(e) - \Delta] - \sum_{e \text{ into } A} \Delta \\ &= \textstyle\sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ into } A} \Delta \\ &\geq c(A, B) - m\Delta \quad \square \end{aligned}$$

Augmentations per Scaling Phase

Proposition

There are at most $2m$ augmentation paths per scaling phase.

Proof.

- Let f be flow at end of previous scaling phase, i.e., with scaling 2Δ
- If f^* is max-flow, then $v(f^*) \leq v(f) + m(2\Delta)$
- Since each augmentation in new phase increases flow by Δ, there can be at most $2m$ augmentations to f $\square$

Removing Dependence on C

- [Edmonds-Karp, Dinitz] Picking augmenting paths with fewest number of edges yields a $O(m^2n)$ algorithm, i.e., independent of C!
- 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 $(A, B) = \{(u, v) \mid u \in A, v \in B\}$

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

Network Flow Facts to Remember

Flow network: directed graph G, capacities c, source s, sink t

- maximum $s-t$ flow can be computed
  - using Ford-Fulkerson algorithm in $O(mC)$ time when capacities are integral and C is an upper bound on the flow
  - using capacity scaling algorithm in $O(m^2 \log C)$ time when capacities are integral
  - using Edmonds-Karp algorithm in $O(m^2 n)$ time when capacities are rational (strongly polynomial time algorithm)
- if capacities are integral then there is a maximum flow that is integral and above algorithms give an integral max flow
- given a flow of value v, can decompose into $O(m+n)$ flow paths of same total value v. integral flow implies integral flow on paths
- maximum flow is equal to the minimum cut and minimum cut can be found in $O(m+n)$ time given any maximum flow