Max Flow Problem

Aidan Johnson and Sam Harris

Linear Programming Final Project

December 3, 2025

Outline

Flow Network Definition

Let $G = (V, E)$ be a directed graph and with source and sink vertices $s, t \in V$.

- Assign each edge $e \in E$ a capacity $c(e)$, the maximum amount of flow that can go across an edge.
- A flow is function that assigns each edge a weight such that:
  - for every $v \in V \setminus \{s, t\}$, the flow into v is equal to the flow out of v
  - the flow on each edge can not exceed the edge capacity

Max Flow Problem Statement

Value of a flow. For a flow f, its value is

$$\mathrm{val}(f) = \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ into } s} f(e).$$

Maximum flow problem. Find a feasible flow f of maximum value among all flows in the network. That is,

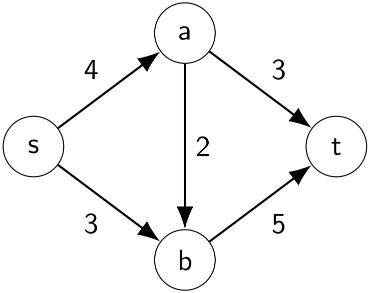
maximize $\mathrm{val}(f)$ subject to $0 \leq f(e) \leq c(e)$ and flow conservation.

Max Flow Problem as an LP

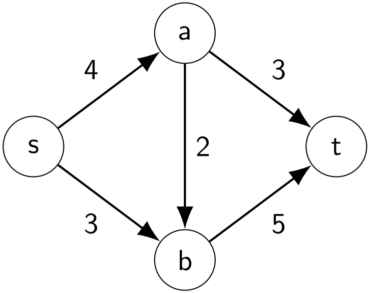
$$\begin{aligned}
\text{maximize} \quad & \sum_{v:(s,v)\in E} f_{sv} \\
\text{s.t.} \quad & \sum_{v:(u,v)\in E} f_{uv} - \sum_{v:(v,u)\in E} f_{vu} = 0, \quad \forall\, u \in V \setminus \{s,t\}, \\
& 0 \le f_{uv} \le c_{uv}, \qquad \forall (u,v) \in E.
\end{aligned}$$

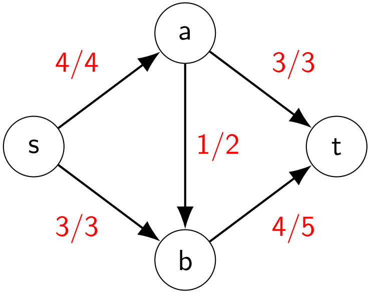
- f_{uv} is the flow from u to v
- Objective function maximizes flow out of s
- First constraint forces a zero net flow on vertices excepting s and t
- Second constraint ensures flow on each edge is nonnegative and does not exceed the edge's capacity

Max Flow Example



Max Flow Example

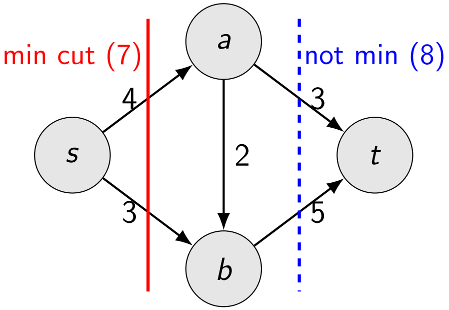




In this example, the maximum flow is 7.

Min Cut Problem Statement

- An $s-t$ cut is a is a node partition (S, T) such that $s \in S$ and $t \in T$.
- The capacity of a cut (S, T) is the sum of the weights of edges **leaving** S.



Min cut problem: Find an $s-t$ cut of minimum capacity

Min Cut Problem as an IP

$$\begin{aligned}
\text{minimize} \quad & \sum_{(u,v)\in E} c_{uv}\, z_{uv} \\
\text{subject to} \quad & z_{uv} \;\geq\; x_v - x_u, \qquad \forall (u,v) \in E, \\
& x_s = 0, \quad x_t = 1, \\
& x_v \in \{0,1\}, \qquad \forall v \in V, \\
& z_{uv} \in \{0,1\}, \qquad \forall (u,v) \in E.
\end{aligned}$$

- c_{uv}: weight of an edge from u to v
- z_{uv}: binary variable equal to 1 if $u \in S$ and $v \in T$, and 0 otherwise
- x_v: binary variable equal to 0 if $v \in S$, and 1 if $v \in T$

Max Flow Min Cut Theorem

Theorem

The value of a max flow is equal to the capacity of a min cut.

Intuition:

- Every flow must go through the smallest 'bottleneck' (capacity of the min cut).
- The max flow uses the full capacity of the min cut.

Some Graph Theory Definitions

Let $G = (V, E)$ be an undirected graph and let $s, t \in V$ be two non-adjacent vertices.

Definition

An **edge cut** of G is a subset of E whose deletion results in a disconnected graph.

Definition

Two paths are **pairwise edge disjoint** if they have no edges in common.

Definition

Let $\kappa'(s, t)$ be the size of the minimum edge-cut such that there is no path between s and t.

Definition

Let $\lambda'(s, t)$ be the maximum number of pairwise edge disjoint paths between s and t.

Menger's Theorem

Theorem

Let $G = (V, E)$ be an undirected graph and let $s, t \in V$ be two non-adjacent vertices. Then $\kappa'(s, t) = \lambda'(s, t)$.

Proof Sketch:

- Form the directed graph G' from G by replacing every edge $uv \in E$ with two directed edges $u \to v$ and $v \to u$. Each edge in G' has capacity 1.
- By definition, $\kappa'(s, t) \geq \lambda'(s, t)$.

Menger's Theorem Proof Sketch

- A minimum capacity cut of G' partitions vertices into parts S and T such that $s \in S$ and $t \in T$. Min-cut(G') = the number of edges from S to T in $G \geq \kappa'(s,t)$.
- By the Max Flow Min Cut Theorem, there exists a max flow f such that val$(f) =$ max-flow$(G') =$ min-cut(G'). If f assigns nonzero flow to two oppositely directed edges, assign both 0 instead which guarantees we only use each edge from G in one direction. There are min-cut(G') many unit flow paths from s to t. These must be disjoint, otherwise we violate the conservation condition. So, f gives min-cut(G') many pairwise edge disjoint paths which implies $\lambda'(s,t) \geq$ max-flow(G').
- $\lambda'(s,t) \geq$ max-flow$(G') =$ min-cut$(G') \geq \kappa'(s,t) \implies \lambda'(s,t) \geq \kappa'(s,t)$.

Greedy Implementation

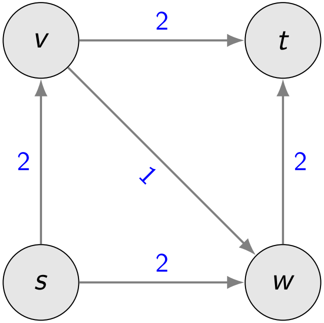
Greedy algorithm:

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \to t$ path P where each edge has open capacity, i.e. $f(e) < c(e)$.
- Augment flow along P.
- Repeat until no such path exists.

Problems with Greedy Implementation

Problem: A bad order of path choices can lead to a sub-maximum flow.

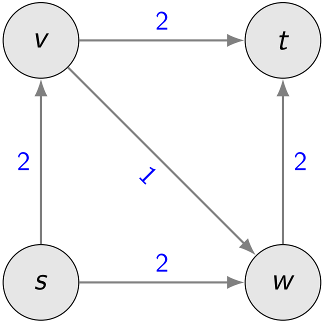
Example: Choosing $s \to v \to w \to t$ as a first path results in a submaximal flow.

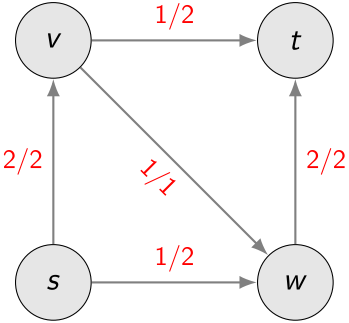


Problems with Greedy Implementation

Problem: A bad order of path choices can lead to a sub-maximum flow.

Example: Choosing $s \to v \to w \to t$ as a first path results in a submaximal flow.





Solution: Residual Flow Network

Residual capacity: For an edge (u, v) with capacity $c(u, v)$ and current flow $f(u, v)$, the *residual capacity* is

$$c_f(u, v) = c(u, v) - f(u, v).$$

Backward edges: If $f(u, v) > 0$, we add a backward edge (v, u) with residual capacity

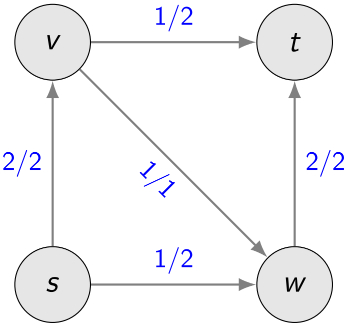
$$c_f(v, u) = f(u, v).$$

The fix:

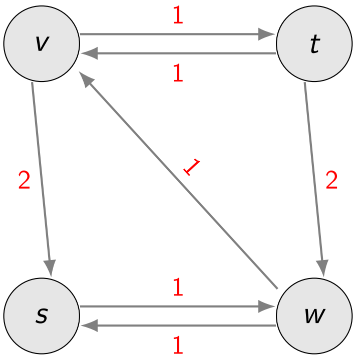
- Forward edges let us send more flow.
- Backward edges let us *undo* previously sent flow.

Residual Flow Example

Submaximal Flow



Residual Flow Network



Ford-Fulkerson Runtime and Implementations

General Ford-Fulkerson Algorithm:

- Start with $f(e) = 0$ for each edge $e \in E$.
- Build residual flow network.
- Find an $s \to t$ path P where each edge has open capacity, i.e. $f(e) < c(e)$ (using BFS or other method).
- Augment flow along P, and update residual flow network.
- Repeat until no such path exists.

Runtime and Implementation

- Generally solves much faster than solving as an LP/IP
- For a graph $G = (V, E)$ with integral capacities at most c, the runtime is $c|V|$.
- Can be improved by carefully choosing augmenting paths based on length and size of bottleneck capacity.
- Fast implementation in networkx library in Python

Applications

Applications of the Max Flow Problem:

- **Network Connectivity:** Finding the maximum number of disjoint paths between two nodes.
- **Bipartite Matching:** Reducing maximum matching problems to a flow network.
- **Circulation Problems:** Optimizing flow with demands and supplies.
- **Project Selection / Scheduling:** Modeling tasks and resource allocation.
- **Image Segmentation:** Graph cuts in computer vision.
- **Transportation / Logistics:** Optimizing traffic, shipping, and pipeline networks.
- **Min-Cut Analysis:** Identifying bottlenecks in networks.

References

Schwartz, R. (2022).
Applications of max flow min cut.
https://www.math.brown.edu/~reschwar/M1230/maxflow.pdf.
Accessed 2025-12-03.

Vanderbei, R. J. (n.d.).
Lecture 2: Linear programming and network flows.
Lecture notes, Princeton University.
Online; accessed via
urlhttps://vanderbei.princeton.edu/542/lectures/lec2.pdf.

Wayne, K. (2004).
Max flow, min cut.
Lecture notes, *COS226: Algorithms and Data Structures*, Princeton University.
Spring 2004 lecture notes.

Wayne, K. D. (2005).
Chapter 7: Network flow i.
https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/07NetworkFlowI.pdf.
Lecture slides for Kleinberg and Tardos, Algorithm Design.

West, D. B. (2001).
Introduction to Graph Theory.
Prentice Hall, Upper Saddle River, NJ, 2 edition.
See Chapter 4.3.