Network Flow

Flow Network

Defn: A flow network is a directed graph

$$G = (V, E)$$

with source $s \in V$ and sink $t \in V$, and all $e \in E$ has capacity $c_e \geq 0$

Feasible Flow

An **s-t flow** is a function $f : E \to \mathbb{R}^+$.

A feasible flow satisfies:

- (i) Capacity constraint: $\forall e \in E : 0 \leq f(e) \leq c_e$
- (ii) Conservation of flow: $\forall v \in V$ except s, t: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$

Residual Network

Defn: Let G be a flow network and f be a flow on G. We define G_f to be the residual network of f on G such that:

- G_f has the same nodes as G
- For each edge $e = (u, v)$ where $f(e) < c_e$, we include $e = (u, v)$ in G_f with a capacity of $c_e - f(e)$ (forward edges)
- For each edge $e = (u, v)$ where $f(e) > 0$, we include $e' = (v, u)$ in G_f with a capacity of $f(e)$ (backward edges)

The **Residual Capacity** is the capacity of an edge in G_f.

Ford-Fulkerson Algorithm

Let P be a simple (does not revisit any node) s-t path in G_f. Define bottleneck(P, f) to be the minimum residual capacity of any edge in P due to f.

Augment is a function that yields a new flow f' in G:

Algorithm 1: augment(f, P)

```
Let b ← bottleneck(P, f);
for each edge (u, v) ∈ P do
    if e = (u, v) is a forward edge then
        increase f(e) in G by b;
    else
        (u, v) is a backward edge, and let e = (v, u);
        decrease f(e) in G by b;
    end
end
return f;
```

Algorithm 2: Ford-Fulkerson(G, s, t)

```
Initially f(e) = 0 for all e in G;
while there is an s-t path in the residual graph G_f do
    Let P be a simple s-t path in G_f;
    f' = augment(f, P);
    Update f to be f';
    Update the residual graph G_f to be G_f';
end
return f;
```

Min-Cut

For any node $v \in V$ in G let us define:

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

$$f^{\text{in}}(v) = \sum_{e \text{ into } v} f(e)$$

Defn: An s-t cut is a partition (S, T) such that:

- $s \in S$
- $t \in T$
- $S \cup T = V$

The **capacity** of (S, T) is the sum of capacities of edges that go from S into T:

$$c(S, T) = \sum_{u \in S,\, v \in T} c_{(u,v)}$$

The **flow** of (S, T) with respect to a flow f is:

$$v(f) = f^{\text{out}}(S) - f^{\text{in}}(S)$$

A **Min-Cut** is a cut with minimum capacity among all s-t cuts.

Max-Flow Min-Cut Theorem

Let G be a flow network with source s and sink t.

$$\begin{aligned} \text{A feasible flow } f \text{ is a max-flow} &\iff \text{there is no } s\text{-}t \text{ path in the residual network } G_f \quad \text{(e.g. no augmenting path in } G_f) \\ &\iff v(f) = c(S, T) \text{ for some min-cut } s\text{-}t \text{ cut } (S, T) \end{aligned}$$

Complexity

Undecidable Decision Problems

There exists undecidable decision problems, such as the Halting problem:

Given a program P and input x, decide whether $P(x)$ terminates.

Class P and NP

P: The class of decision problems solvable in polynomial time.
NP: The class of decision problems whose solutions can be verified in polynomial time.
Defn: A decision problem $f \in NP$ if there exists a program $A \in P$ such that:

$$f(x) = 1 \iff \exists \text{ witness } w \in \{0, 1\}^* \text{ such that } A(x, w) = 1$$

Karp Reduction

Defn: Problem f polynomial Karp-reduces to g if there exists an algorithm $R \in P$ such that:

$$f(x) = 1 \iff g(R(x)) = 1$$

And we write $f \leq_m g$

NP-Complete

Defn: A problem g is NP-complete if $\forall f \in NP$:

$$f \leq_m g$$

Cook-Levin Theorem

The Boolean Satisfiability Problem (SAT) is NP-complete.