Flows (recap)

Optimization II: Max Flows → LPs

Network $G = (V, E)$ directed graph, source $s$, sink $t$

Capacities $\text{Cap} : E \to \mathbb{R}^+$

Flow: map $f : E \to \mathbb{R}^+$ s.t.

1. $0 \leq \text{Cap}(e)$
2. Flow conservation $\forall u \neq s, t$,

$$\sum_{e \in \text{into } u} f(e) = \sum_{e \in \text{out of } u} f(e).$$

Value $\text{Val}(f) = \sum_{e} f(e)$

(assume no edge into $s$, no edges out of $t$)

Max Flow: maximize $\text{Val}(f)$ for flow $f$.

Thm: if $G$ has integer (non-negative) capacities then

Ford-Fulkerson Algorithm finds max flow in time $O(mf^*)$ $f^* = \text{value of max flow}$.

Fact: with integer capacities, $f^*$ finds integer-valued flows

$\Rightarrow$ An integer-valued max flow if capacities are integer.

Hence: if you can show that $f$ flow with value $V$ that is fractional

$\Rightarrow$ Also a flow $f'$ with value $\geq V$ that is integer-valued if capacities of edges are integral.
Network flows are important modeling language. Models problems like:

- Job scheduling
- Airplane scheduling
- Baseball elimination
- Project selection

See previous lectures for definitions. (Or Kleinberg & Tardos)

- Finds integer flows when capacities are integers

- Ford-Fulkerson takes $O(mF^*)$ time

- Faster algorithms exist (Dinic's, Edmonds-Karp, etc.).

Today:

- Max flow = Min cut

- Min cost $\geq$ Max Flows.

- Leads into linear programming
MaxFlow = Min Cut

Recall: a cut in the graph G is a bi-partition of the vertex set into \((A, B = V \setminus A)\) such that \(s \in A\) and \(t \in B\).

\[
\text{Capacity (cut } A, B) = \text{ total capacity of edges going from } A \text{ to } B.
\]

Fact: any flow from \(s \rightarrow t\) has value \( \leq \) capacity of any \(s-t\) cut \(A, B\).

\[
\Rightarrow \text{Max flow value } \leq \text{min- } s-t \text{-cut capacity}
\]

\(\text{Weak duality}\)

Thm: Max flow = Min cut.

\(\text{strong duality}\)

For a max flow E a simple "proof" that it is optimal, i.e. a cut whose capacity certifies that no more flow can be sent across from \(s\) to \(t\).

PF: Run FF. This keeps sending flow until residual graph has no more paths from \(s \rightarrow t\) with positive capacity. Say we can reach some set \(A\) of vertices from source \(s\), using edges of positive capacity.

We claim that \(\text{cap}(A, \overline{A} = B)\) is equal to current flow value in the original graph \(G\).

Proof by induction on FF: If we push \(x\) amounts of flow from \(s \rightarrow t\) then residual capacity on this cut falls by \(x\). This will prove that:

\[
\text{Initially: } \text{cap}(A, B) = C \text{ (say) } \Rightarrow \text{flow in hand} = 0.
\]

\[
\text{Finally: } \text{cap}(A, B) = 0 \text{ because we haven't } \Rightarrow \text{flow in hand must be } x.
\]
OK, so to prove that pushing a flow from s to t reduces the capacity of the cut (A,B) by x.

(say push on a path, since that is what FF does)

Path must cross an odd # 4 times. (Say 3 times, same idea for all odds)

All forward edges lost x units of capacity.
But all backwards edges gained x.

We crossed A-B boundary (2k+1) times for some k.
⇒ lost (k+1)x capacity in new residual graph
    gained kx
⇒ cap(A,B) ↓ by x.

This completes the proof that Max Flow = Min Cut

Special case of general phenomenon,
for many optimization problems (especially convex optimization) "duals" give
lots of information about optimality.
Topic II: Faster Flow Algorithms

Ford-Fulkerson Says: — pick any path and push flow on it.
Build new residual graph. Repeat.

Which path? Makes difference.

(I) Shortest path that has non-zero capacity?
(II) "Fattest" path?

Both make algorithm much faster.

FF takes $O(mF^2)$ time $\leftarrow$ not polynomial time

but Shortest Path FF takes $O(m^2n)$ time
Fattest Path FF takes $O(m^2 \log F)$ time

Exist other faster flow algorithms

- active area of research

- see animations on webpage, details in books.

Now each edge \( e \) has 2 numbers \( \text{cap}(e) \) and \( \text{cost}(e) \).

Want \( s-t \) flow that:

(a) \( \text{value}(\text{flow } f) = \text{max-flow, value} \)

(b) \( \text{cost}(\text{flow}) = \sum_e f(e) \text{ cost}(e) \) is minimized.

E.g. in airline scheduling example, what if want the cheapest way to schedule all the \( m \) flights.

Thm: The min-cost max-flow can be found efficiently.

Moreover: if capacities are integers, then \( \exists \) a min-cost-max-flow

that sends integer amount of flow \( f(e) \) on every edge \( e \).

This allows us to use it for

scheduling / assignment problems.

Here's the algo:

Run FF-like algo.

- Each time find a shortest path (according to cost) from \( s \to t \).
- Send flow on it.
- Create residual graph. But "back" edges have negative cost.

\[
\begin{align*}
\text{cap}, \text{cost} & \quad \to \quad \text{cap} - x, \text{cost} \\
\text{push } x \text{ flow} & \quad \Rightarrow \\
\text{"get moneyback" if undo the flow}.
\end{align*}
\]

\( \text{will not give proof.} \)

Optional: checkout text / notes from 451.