

This and other notes are available from <http://www.cs.berkeley.edu/~yozo/cs170.fa05/>

Network Flows

- We let $f_{u,v}$ be the *net* flow from u to v . In particular, $f_{u,v} = -f_{v,u}$ for all $u, v \in V$. We require $f_{u,v} \leq c_{u,v}$. Conservation of flow requires $\sum_v f_{u,v} = 0$ for any vertex $u \neq s, t$.
- Cuts. Any subset $S \subset V$ of nodes such that $s \in S$ but $t \notin S$ defines a cut, splitting the vertices into two disjoint sets S and $T = V \setminus S$. We can consider the net flow from S to T :

$$f(S, T) = \sum_{u \in S, v \in T} f_{u,v}.$$

Does this take into account “backward” flows going from a node in T to a node in S ? Verify that $f(X, X) = 0$, $f(X, Y) = -f(Y, X)$, and $f(X, Y \cup Z) = f(X, Y) + f(X, Z)$ provided Y and Z are disjoint ($Y \cap Z = \emptyset$). What does $|f| = f(\{s\}, V - \{s\})$ represent?

- Cut capacity. For any cut S, T we can also consider the capacity of that cut:

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

- Ford-Fulkerson algorithm: find augmenting path from s to t , construct residual network, repeat.
- Edmonds-Karp algorithm: use BFS to find the augmenting paths. Gives an $O(VE^2)$ algorithm.
 - At each iteration the shortest path from u to v (in residual network) strictly increases. (Why? This requires some thought).
 - When we augment a flow through a path p , we say that edge e becomes critical if $f(e) = c(e)$. Each edge $e = (u, v)$ can become critical $O(V)$ times. (Why?)
 - The above two points says that there can be at most $O(VE)$ critical edges during the execution of the algorithm, leading to $O(VE^2)$ algorithm.

Currently best network flow algorithm is a preflow-push algorithm that runs in $O(VE \lg(V^2/E))$ time.

- Weak duality: $c(S, T) \leq f(S, T)$ for any cut. Why?
- Strong duality: there exists a cut S, T such that $c(S, T) = f(S, T)$. The minimum capacity cut has the same capacity as the maximum flow. This follows from Ford-Fulkerson algorithm (Why?)

Duality

Minimization of $c^T x$ subject to $Ax \geq b$ and $x \geq 0$ is equivalent to maximization of $b^T y$ subject to $A^T y \leq c$ with $y \geq 0$.

Linear Program: Max-Flow vs. Min-Cut

- Max-flow: maximize $\sum_{v \in V} f_{s,v}$, subject to

$$\begin{aligned} f_{u,v} &\leq c_{u,v} && \text{for all } (u,v) \in E \\ \sum_{(v,u) \in E} f_{v,u} - \sum_{(u,v) \in E} f_{u,v} &= 0 && \text{for each } u \end{aligned}$$

and $f_{u,v} \geq 0$. If we introduce slack variables $g_u \geq 0$ then the second set of constraint becomes

$$\sum_{(v,u) \in E} f_{v,u} - \sum_{(u,v) \in E} f_{u,v} - g_u \leq 0$$

- Min-cut. Consider any cut S, T . For any $u, v \in V$, let

$$x_u = \begin{cases} 1 & \text{if } u \in S \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Note that set of values of x_u determines the set S and T . We also let

$$y_{uv} = \begin{cases} 1 & \text{if } u \in S \text{ and } v \in T \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

We want to minimize $c = \sum_{u,v \in V} c_{u,v} y_{uv}$. Assuming the capacities $c(u,v)$ are non-negative, this means that we want to make each y_{uv} as small as possible.

For each variable y_{su} we have the constraint $y_{su} + x_u \geq 1$ to make sure $y_{su} = 1$ iff $x_u = 0$. Similarly for each variable y_{ut} we have the constraint $y_{ut} - x_u \geq 0$ to make sure $y_{ut} = 1$ iff $x_u = 1$. Finally, for each variable y_{uv} where $u \neq s$ and $v \neq t$ we have the constraint $y_{uv} - x_u + x_v \geq 0$. This forces $y_{uv} = 1$ only if $x_u = 1$ and $x_v = 0$ (Why, and why do we want this?) Thus unless $y_{uv} \geq 1$ is forced, $y_{uv} = 0$.

This means that the linear program above describes the MIN-CUT problem, and that it is (roughly) a dual of the MAX-FLOW linear program. Notice the MAX-FLOW has $|E| + |V| - 2$ constraints and $|E|$ variables, while MIN-CUT has $|E| + |V| - 2$ variables and $|E|$ constraints.