CS 170 Discussion 8 Reference Sheet

Simplex:

1. Start at an arbitrary vertex.
2. Denote the current vertex as x.
3. Look at all neighboring vertices y to x.
4. Find the neighbor y^* with the best objective value (if the LP is minimizing, y^* should have the smallest objective value, etc.).
5. If y^* has a better value than x, then we move to (i.e. visit) y^* and repeat steps 2 to 4. Otherwise, we have found the optimizer of the LP, and simply return x.

↪ Runtime: worst case exponential time, average case polynomial time.

LP Solver Runtime: Any LP can be solved in (worst case) polynomial time using the Ellipsoid or Interior Point Method (IPM). *Note: you do not need to know how the Ellipsoid method or IPM work, but you should know that they can be used to solve an LP in polynomial time.*

Flow. The *capacity* indicates how much flow can be allowed on an edge. Given a directed graph $G = (V, E)$ with edge capacites $c(u, v)$ (for all $(u, v) \in E$) and vertices $s, t \in V$, a flow is a mapping $f : E \to \mathbb{R}^+$ that satisfies

- **Capacity constraint:** $f(u, v) \leq c(u, v)$, the flow on an edge cannot exceed its capacity.
- **Conservation of flows:** $f^{\text{in}}(v) = f^{\text{out}}(v)$, flow in equals flow out for any $v \notin \{s, t\}$

Here, we define $f^{\text{in}}(v) = \sum_{u:(u,v)\in E} f(u, v)$ and $f^{\text{out}}(v) = \sum_{u:(v,u)\in E} f(u, v)$. We also define $f(v, u) = -f(u, v)$, and this is called *skew-symmetry*. Note that the total flow in the graph is $\sum_{v:(s,v)\in E} f(s, v) = \sum_{u:(u,t)\in E} f(u, t)$, where s is the source node and t is the target node.

Residual Graph. Given a flow network (G, s, t, c) and a flow f, the *residual capacity* (w.r.t. flow f) is denoted by $c_f(u, v) = c_{uv} - f_{uv}$. And the *residual network* $G_f = (V, E_f)$ where $E_f = \{(u, v) : c_f(u, v) > 0\}$.

Max Flow. Given a directed graph $G = (V, E)$ with edge capacities $c(u, v)$ for all $(u, v) \in E$ and vertices $s, t \in V$, the goal is to compute the maximum flow that can go from s to t. This can be solved with either Ford-Fulkerson (or its optimized variant Edmonds-Karp).

Ford-Fulkerson. Keep pushing along $s - t$ paths in the residual graph and update the residual graph accordingly. The runtime of this algorithm is $O(mF)$, where $m = |E|$ and F is the value of the max flow.

Edmonds-Karp. We can implement Ford-Fulkerson using BFS to find the $s-t$ paths. This yields a faster runtime of $O(nm^2)$, where $n = |V|, m = |E|$.

Min-Cut. Given a directed graph $G = (V, E)$ with edge weights $w(u, v)$ for all $(u, v) \in E$ and vertices $s, t \in V$, the goal is to compute the lightest $s - t$ cut (i.e. the cut separating t from s with the smallest sum of edge weights crossing it). Note that the Min Cut problem is the dual of the Max Flow problem.