Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. They are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

Canonical Form. The canonical form of a linear program is

$$\begin{aligned} &\text{maximize } c^\top x \\ &\text{subject to } Ax \leq b \\ &\qquad x \geq 0 \end{aligned}$$

where $x \geq 0$ means that every entry of the vector x is greater than or equal to 0. Here, $f(x) = c^\top x$ is the **objective function**, and $Ax \leq b, x \geq 0$ are the **constraints**. The **feasible region** is the intersection of all the halfspaces defined by the constraints.

Dual. The dual of the canonical LP is

$$\begin{aligned} &\text{minimize } y^\top b \\ &\text{subject to } y^\top A \geq c^\top \\ &\qquad y \geq 0 \end{aligned}$$

Weak duality: The objective value of any feasible primal $\leq$ objective value of any feasible dual. In other words, if we define x^* to be the optimizer of the LP above (in canonical form) and y^* to be the optimizer of its dual, weak duality implies that

$$c^\top x^* \leq (y^*)^\top b$$

Strong duality: The *optimal* objective values of a primal LP and its dual are equal. In other words,

$$c^\top x^* = (y^*)^\top b$$

Note that strong duality always holds for linear programs as long as the primal or dual are feasible.

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.

$\hookrightarrow$ 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 know that they can be used to solve an LP in polynomial time.*

1 Simply Simplex

Consider the following linear program.

$$\max\ 5x + 4y$$
$$\text{subject to } \begin{cases} x + 3y \leq 15 \\ 3x - 2y \leq 12 \\ 4x + 5y \geq 16 \\ x \geq 0, y \geq 0 \end{cases}$$

(a) Sketch the feasible region.

(b) Simulate the Simplex algorithm on this LP. What are the vertices visited?

2 Job Assignment

There are I people available to work J jobs. The value of person i working 1 day at job j is a_{ij} for $i = 1, \ldots, I$ and $j = 1, \ldots, J$. Each job is completed after the sum of the time of all workers spend on it add up to be 1 day, though partial completion still has value (i.e. person i working c portion of a day on job j is worth $a_{ij}c$). The problem is to find an optimal assignment of jobs for each person for one day such that the total value created by everyone working is optimized. No additional value comes from working on a job after it has been completed.

(a) What variables should we optimize over? I.e. in the canonical linear programming definition, what is x?

(b) What are the constraints we need to consider? Hint: there are three major types.

(c) What is the maximization function we are seeking?

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 of the graph 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.

3 Max-Flow Min Cut Basics

For each of the following, state whether the statement is True or False. If true provide a short proof, if false give a counterexample.

(a) If all edge capacities are distinct, the max flow is unique.

(b) If all edge capacities are distinct, the min cut is unique.

(c) If all edge capacities are increased by an additive constant, the min cut remains unchanged.

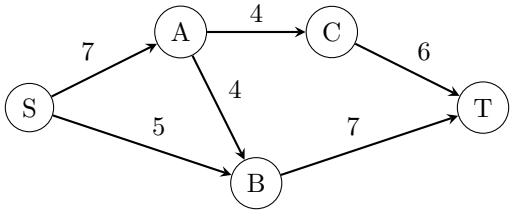
(d) If all edge capacities are multiplied by a positive integer, the min cut remains unchanged.

(e) In any max flow, there is no directed cycle on which every edge carries positive flow.

(f) There exists a max flow such that there is no directed cycle on which every edge carries positive flow.

4 Residual in graphs

Consider the following graph with edge capacities as shown:



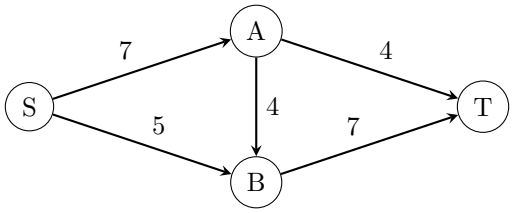
(a) Consider pushing 4 units of flow through $S \to A \to C \to T$. Draw the residual graph after this push.

(b) Compute a maximum flow of the above graph. Find a minimum cut. Draw the residual graph of the maximum flow.

5 Max Flow, Min Cut, and Duality

In this exercise, we will demonstrate that LP duality can be used to show the max-flow min-cut theorem.

Consider this graph instance of max flow:



Let f_1 be the flow pushed on the path $\{S, A, T\}$, f_2 be the flow pushed on the path $\{S, A, B, T\}$, and f_3 be the flow pushed on the path $\{S, B, T\}$. The following is an LP for max flow in terms of the variables f_1, f_2, f_3:

$$
\begin{aligned}
\max \quad & f_1 + f_2 + f_3 & \\
& f_1 + f_2 \leq 7 & \text{(Constraint for } (S, A)) \\
& f_3 \leq 5 & \text{(Constraint for } (S, B)) \\
& f_1 \leq 4 & \text{(Constraint for } (A, T)) \\
& f_2 \leq 4 & \text{(Constraint for } (A, B)) \\
& f_2 + f_3 \leq 7 & \text{(Constraint for } (B, T)) \\
& f_1, f_2, f_3 \geq 0 &
\end{aligned}
$$

The objective is to maximize the flow being pushed, with the constraint that for every edge, we can't push more flow through that edge than its capacity allows.

(a) Find the dual of this linear program, where the variables in the dual are x_e for every edge e in the graph.

(b) Consider any cut in the graph. Show that setting $x_e = 1$ for every edge crossing this cut and $x_e = 0$ for every edge not crossing this cut gives a feasible solution to the dual program.

(c) Based on your answer to the previous part, what problem is being modelled by the dual program? By LP duality, what can you argue about this problem and the max flow problem?

6 Flow vs LP

You play a middleman in a market of m suppliers and n purchasers. The i-th supplier can supply up to $s[i]$ products, and the j-th purchaser would like to buy up to $b[j]$ products.

However, due to legislation, supplier i can only sell to a purchaser j if they are situated at most 1000 miles apart. Assume that you're given a list L of all the pairs (i, j) such that supplier i is within 1000 miles of purchaser j. Given m, n, $s[1..m]$, $b[1..n]$, and L as input, your job is to compute the maximum number of products that can be sold. The runtime of your algorithm must be polynomial in m and n.

For parts (a) and (b), assume the product is divisible—that is, it's OK to sell a fraction of a product.

(a) Show how to solve this problem, using a network flow algorithm as a subroutine. Describe the graph and explain why the output from the network flow algorithm gives a valid solution to this problem.

(b) Formulate this as a linear program. Explain why this correctly solves the problem, and the LP can be solved in polynomial time.

(c) Now let's assume you *cannot* sell a fraction of a product. In other words, the number of products sold by each supplier to each purchaser must be an integer. Which formulation would be better, network flow or linear programming? Explain your answer.