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

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

where $x \ge 0$ 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 $Ax \le b, x \ge 0$ 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

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

Weak duality: The objective value of any feasible primal $\le$ 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

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

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.

↪ 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.

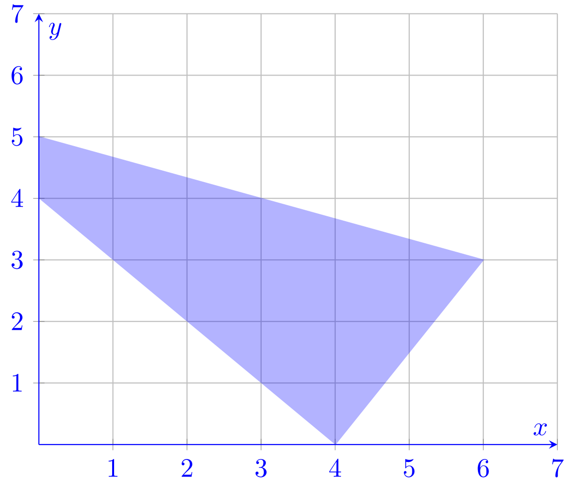
$$\max 5x + 4y$$
$$\text{subject to } \begin{cases} x + 3y \leq 15 \\ 3x - 2y \leq 12 \\ 4x + 4y \geq 16 \\ x \geq 0, y \geq 0 \end{cases}$$

(a) Sketch the feasible region.

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

Solution:

(a) We sketch the feasible region below:



(b) The vertices of the feasible region are $(0, 4)$, $(0, 5)$, $(6, 3)$, $(4, 0)$. Depending on the starting vertex, Simplex will visit the vertices in a greedy fashion that minimizes the objective $5x + 4y$. For instance, suppose we started at $(0, 4)$. Here are the steps that Simplex would take:

(a) We note that the objective value at $(0, 4)$ is $5(0) + 4(4) = 16$. Its neighbors are $(0, 5)$ and $(4, 0)$, with objective values $5(0) + 4(5) = 20$ and $5(4) + 4(0) = 20$. Since we're trying to maximize the objective function, we know that we should go to one of the neighbors because their objective values are higher. Generally we go to the neighbor with the higher objective value, but in this it doesn't really matter since they have the same value. Let's say we go to $(0, 5)$.

(b) From before, we know that the objective value of $(0, 5)$ is 20. Now, we compute its neighbors' objective values: we already know that the objective value of $(0, 4)$ is 16, so now just need to find the objective value of $(6, 3)$, which is $5(6) + 4(3) = 42$. We see that $42 > 20$, and so we move to $(6, 3)$.

(c) The objective value of $(6, 3)$ is 42, which is higher than the objective values of any of its neighbors (the value of $(0, 5)$ is 20 and the value of $(4, 0)$ is also 20). Thus, we have found the optimum point in the LP!

Throughout this run of Simplex, we visited $(0, 4)$, $(0, 5)$, and $(6, 3)$ in that order.

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?

Solution: An assignment x is a choice of numbers x_{ij} where x_{ij} is the portion of person i's time spent on job j.

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

Solution: First, no person i can work more than 1 day's worth of time.

$$\sum_{j=1}^{J} x_{ij} \leq 1 \qquad \text{for } i = 1, \ldots, I.$$

Second, no job j can be worked past completion:

$$\sum_{i=1}^{I} x_{ij} \leq 1 \qquad \text{for } j = 1, \ldots, J.$$

Third, we require positivity.

$$x_{ij} \geq 0 \qquad \text{for } i = 1, \ldots, I, j = 1, \ldots, J.$$

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

Solution: By person i working job j for x_{ij}, they contribute value $a_{ij}x_{ij}$. Therefore, the net value is

$$\sum_{i=1,j=1}^{I,J} a_{ij}x_{ij} = A \bullet x.$$

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.

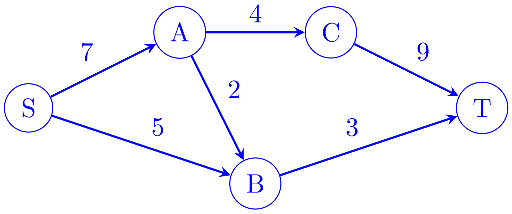
(d) If all edge capacities are multiplied by a positive integer, the min cut remains unchanged.

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

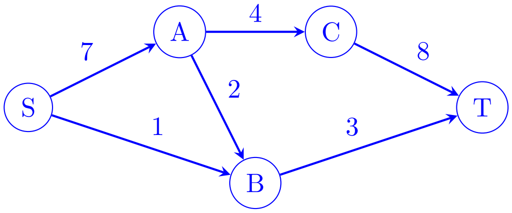
(f) There exists a max flow such that there is no directed cycle on which every edge carries positive flow.

Solution:

(a) False. Consider the following graph:



(b) False. Consider the graph below. The cut SA and the cut SAB are both size 7.

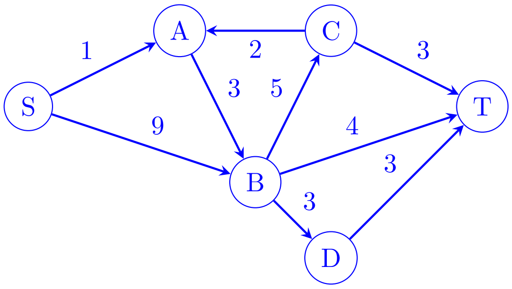


(c) False. Add one to all of the edge capacities of the graph in part (b). The cut SA and the SAB have different values now.

(d) True. Let the value of a cut be $\sum_e c_e$ and the value of the minimum cut be $\sum_{e'} c_{e'}$. The minimum cut must still be the minimum cut after multiplying the edges by a positive constant, due to the distributive property:

$$a\sum_{e'} c_{e'} - a\sum_{e} c_e = a(\sum_{e'} c_{e'} \sum_{e} c_e)$$

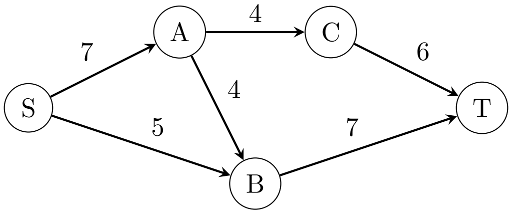
(e) False. Consider the graph below:



(f) True. See any graphs other than (e). Consider the graph in part (e) without the edge SA.

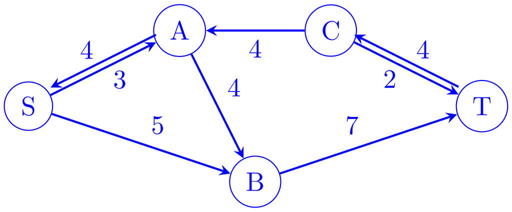
4 Residual in graphs

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.

Solution:

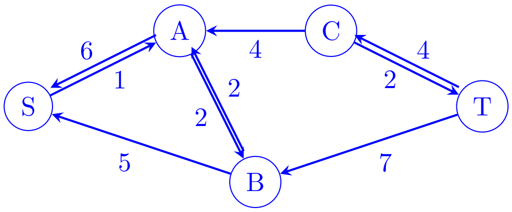


(b) Compute a maximum flow of the above graph. Find a minimum cut. Draw the residual graph of the maximum flow.

Solution: A maximum flow of value 11 results from pushing:

- 4 units of flow through $S \to A \to C \to T$;
- 5 units of flow through $S \to B \to T$; and
- 2 units of flow through $S \to A \to B \to T$.

(There are other maximum flows of the same value, can you find them?) The resulting residual graph (with respect to the maximum flow above) is:

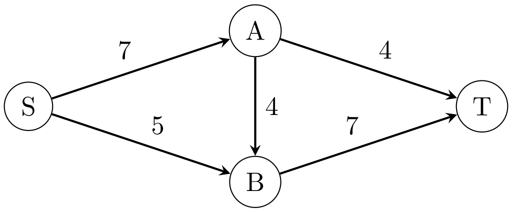


A minimum cut of value 11 is between $\{S, A, B\}$ and $\{C, T\}$ (with cross edges $A \to C$ and $B \to T$).

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?

Solution:

(a) The dual is:

$$\begin{aligned} \min \quad & 7x_{SA} + 5x_{SB} + 4x_{AT} + 4x_{AB} + 7x_{BT} \\ x_{SA} + x_{AT} \geq 1 & \qquad \text{(Constraint for } f_1\text{)} \\ x_{SA} + x_{AB} + x_{BT} \geq 1 & \qquad \text{(Constraint for } f_2\text{)} \\ x_{SB} + x_{BT} \geq 1 & \qquad \text{(Constraint for } f_3\text{)} \\ x_e \geq 0 & \quad \forall e \in E \end{aligned}$$

(b) Notice that each constraint contains all variables x_e for every edge e in the corresponding path. For any $s - t$ cut, every $s - t$ path contains an edge crossing this cut. So for any cut, the suggested solution will set at least one x_e to 1 on each path, giving that each constraint is satisfied.

(c) The dual LP is an LP for the min-cut problem. By the previous answer, we know the constraints describe solutions corresponding to cuts. The objective then just says to find the cut of the smallest size. By LP duality, the dual and primal optima are equal, i.e. the max flow and min cut values are equal.

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.

Solution:

(a) *Algorithm:* We create a bipartite graph with $m + n + 2$ nodes. Label two of the nodes as a "source" and a "sink." Label m nodes as suppliers, and n nodes as purchasers. Now, we will create the following edges:

- Create an edge from the source to supplier i with capacity $s[i]$.
- For each pair (i, j) in L, create an edge from supplier i to purchaser j with infinite capacity.
- Create an edge from purchaser j to the sink with capacity $b[j]$. We then plug this graph into our network flow solver, and take the size of the max flow as the number of dollars we can make.

Proof of correctness: We claim that the value of the max flow is precisely the maximum amount transactions we can make. To show this, we can show that a strategy of selling products corresponds exactly to a flow in this graph and vice versa.

For any flow, let $x_{i,j}$ be the amount of flow that goes from the node of supplier i to the node of purchaser j. Then, we claim a product selling strategy will sell exactly $x_{i,j}$ products from supplier i to purchaser j. This is a feasible strategy, since the flow going out of the node of supplier i is already bounded by $s[i]$ by the capacity of the edge from the source to that node, and similarly for the purchasers. Similarly, for the other direction, one can observe that any feasible selling strategy leads to a feasible flow of the same value.

(b) We define a variable $x_{i,j}$, denoting the amount of products we take from supplier i and sell to purchaser j. Then, we have the following linear program

$$\begin{aligned}
\max \sum_{i=1}^{m}\sum_{j=1}^{n} x_{i,j} \\
\sum_{i=1}^{m} x_{i,j'} \leq b[j'], &\text{ for all } j' \in [1, n] \\
\sum_{j=1}^{n} x_{i',j} \leq s[i'], &\text{ for all } i' \in [1, m] \\
x_{i,j} = 0, &\text{ for all } (i, j) \notin L \\
x_{i,j} \geq 0, &\text{ for all } (i, j)
\end{aligned}$$

The linear program has mn variables and $O(mn)$ linear inequalities, so it can be solved in time polynomial in m and n.

(c) Network flow is better. Linear programming is not guaranteed to find an integer solution (not even if one exists), so the approach in part (b) might yield a solution that would involve selling fractional products. In contrast, since all the edge capacities in our graph in part (a) are integers, the Ford-Fulkerson algorithm for max flow will find an integer solution. Thus, max flow is the better choice, because there are algorithms for that formulation that will let us find an integer solution.