CS 255: Homework 2

Due October 6 at 11:59 pm

Instructions:

1. You may work in groups of up to 3 students.
2. Each group makes one submission on Canvas.
3. Include the name and SJSU ids of all students in the group.
4. You may discuss high level concepts with other groups, but do not copy other's work.

Problem 1

The Autocratic Party is gearing up their fund raising campaign for the 2024 election. Party leaders have already chosen their candidates for president and vice-president, as well as various governors, senators, representatives, judges, city council members, etc. For each candidate, the party leaders have determined how much money they must spend on that candidate's campaign to guarantee their election.

The party is soliciting donations from each of its members. Each voter has declared the total amount of money they are willing to give each candidate between now and the election. (Each voter pledges different amounts to different candidates. For example, everyone is happy to donate to the presidential candidate, but most voters in San Jose will not donate anything to the candidate for Trash Commissioner of Chicago). Federal election law limits each person's total political contributions to $100 per day.

Describe and analyze an algorithm to compute the donation schedule, describing how much money each voter should send to each candidate on each day, that guarantees that every candidate gets enough money to win their election. The schedule must obey federal laws and individual voters' budget constraints. If no such schedule exists, your algorithm should report that fact.

Assume that there are n candidates, p party members, and d days until the election. The input to your algorithm is a pair of arrays $Win[1 \dots n]$ and $Limit[1 \dots p][1 \dots n]$, where $Win[i]$ is the amount of money candidate i needs to win, and $Limit[i][j]$ is the total amount of party member i is willing to donate to candidate j.

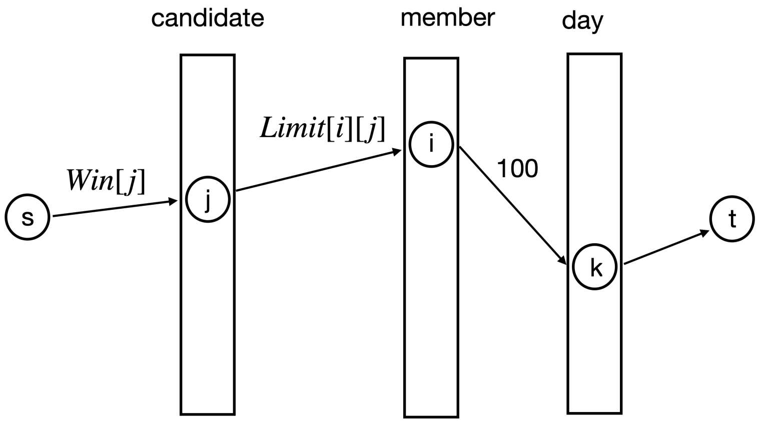
Your algorithm should return an array $Donate[1 \dots p][1 \dots n][1 \dots d]$, where $Donate[i][j][k]$ is the amount of money party member i should donate to candidate j on day k.

Solution: We follow the approach outlined in lecture 6. There are three discrete sets that interact with each other: candidates, party members, and days. Specifically, there are two types of connections:

- For each member-candidate pair (i, j), $Limit[i][j]$ gives the total amount member i is willing to give to candidate j. We add an edge connecting i and j with capacity $Limit[i][j]$.

- Each member can donate no more than \$100 dollars per day. We make edge between member i and day k with capacity 100.

We can organize these sets in the order: candidate, member, day. Since candidate j needs $Win[j]$ total donations to win, we create a edge from the source s to candidate j with capacity $Win[j]$. We also create edges connecting each day k to the sink t with capacity ∞. Refer to the figure below for an illustration of the construction.



Any feasible flow on this graph can be decomposed in to s-t paths where each path defines a member, candidate, day tuple (i, j, k). The minimum flow over any edge in a given path can be interpreted as the amount member i donates to candidate j on day k. The capacity constraints between candidates, members, and days ensure that donations obey members' daily contribution limits, and member limits on contributions to each candidate. If the flow saturates the edges leaving the source, then there is a feasible donation schedule which gives each candidate the total amount of money needed to win their election.

We find a donation schedule by computing a max-flow and decomposing the flow into paths as described above. The total number of vertices in this network is $n+p+d+2$. There are $n + np + pd + d$ edges. We can find a max-flow in the network in $O(VE) = O\big((n + p + d)(n + np + pd + d)\big) = O((n + p + d)(np + pd))$ time. Flow decomposition also takes $O(VE)$ time.

Problem 2

Every year, the SJSU Computer Science Department assigns professors to various faculty committees. There are n faculty member and c committees. Each faculty member has submitted a list of their *prices* for serving on each committee; each price could be positive, negative, zero, or even infinite. For example, a professor might say that she would serve on the Student Recruitment Committee for \$ 100 dollars, that she would *pay* \$200 dollars to serve on the CS 255 Course Revision Committee, and that she would not serve on the Party Planning Committee for any price.

The department knows how many professors are needed for each committee, and has complied a list of professors who would be suitable members for each committee. If the department assigns an instructor to a committee, then they must pay that professor's price from the department budget. (Negative prices increase the department's budget.)

The CS Department needs to assign professors to committees so that: (a) each committee is full, (b) only suitable and willing professors are assigned to each committee, (c) no professor is assigned to more than three committees, and (4) the total cost of the assignment is as small as possible. Describe and analyze an efficient algorithm that either solves this problem, or correctly reports that there is no valid assignment whose total cost in finite.

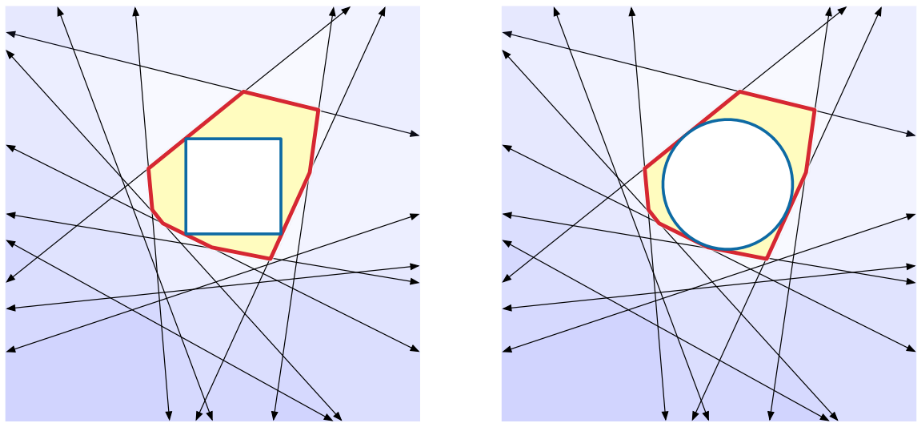
Solution: We model the problem as a min-cost flow by defining the following graph. We create a vertex for each committee and each professor. We add a edge between committee i and professor j with cost equal to the professor i's price for serving on committee j. We don't connect a professor to a committee if their price is infinite or the professor is not suitable for that committee. These edges have capacity 1. We also add edges between the source and each committee with no cost and capacity equal to the number of professors needed for that committee. We add edges from professors to the sink with no cost and capacity equal to 3 (the number of committees a professor can serve on). Let d equal the total number of positions on all the committees. Then, we find the min-cost flow sending d units of flow form s to t. If there is no feasible way to send d units of flow, then no valid assignment exists. The number of vertices is $n + c + 2$, and the number of edges is $n + nc + c$.

Problem 3

A set of n linear inequalities of the form $a_i x + b_i y \leq c_i$, describe a convex polygon P in $\mathbb{R}^2$. Design linear programs to solve the following problems.

(a) Find the largest square with horizontal and vertical sides that lies entirely in P.

(b) Find the largest circle that lies entirely inside of P.



Solution:
(a). We can define the square using the location of the bottom-left corner $p_0 = (x_0, y_0)$ and its side length s. The remaining corners are points $p_1 = (x_0 + s, y_0)$, $p_2 = (x_0, y_0 + s)$, and

$p_3 = (x_0 + s, y_0 + s)$. Since the polygon is convex, it is enough to check that each corner satisfies the n inequalities describing P. This follows since a convex set contains the line connecting any two points in the set. Therefore, we can choose the location of the bottom-left corner and maximize the side length s such that all four corners remain in the polygon.

$$\begin{aligned} & \max s \\ \text{s.t.} \quad & a_i x_0 + b_i y_0 \le c_i, \ \forall i \\ & a_i(x_0 + s) + b_i y_0 \le c_i, \ \forall i \\ & a_i x_0, + b_i(y_0 + s) \le c_i, \ \forall i \\ & a_i(x_0 + s) + b_i(y_0 + s) \le c_i, \ \forall i \\ & s \ge 0, \ x_0, y_0 \in \mathbb{R}. \end{aligned}$$

There are 3 decision variables and $O(n)$ constraints.

(b). Original solution was not correct. Everyone received full credit for this problem.

Let (x_0, y_0) be the center of the circle, and r be its radius. For each inequality defining P, $a_i x + b_i y \le c_i$, we need to ensure that the distance from the center (x_0, y_0) to the line $a_i x + b_i y = c_i$ is less than or equal to the radius r. Consider a point on the line through the center in the direction of the i'th constraint

$$(x_0 + a_i x, \ y_0 + b_i y).$$

Such a point is at a distance of r from the center (x_0, y_0) when

$$\left(x_0 + \frac{a_i r}{\sqrt{a_i^2 + b_i^2}}, \ y_0 + \frac{b_i r}{\sqrt{a_i^2 + b_i^2}} \right).$$

This point satisfies the i'th constraint $a_i x + b_i y \le c_i$ if

$$\begin{aligned} c_i &\ge a_i\left(x_0 + \frac{a_i r}{\sqrt{a_i^2 + b_i^2}} \right) + b_i\left(y_0 + \frac{b_i r}{\sqrt{a_i^2 + b_i^2}} \right) \\ &= a_i x_0 + b_i y_0 + \sqrt{a^2 + b^2} r. \end{aligned}$$

This leads to the following linear program:

$$\begin{aligned} & \max r \\ \text{s.t.} \quad & a_i x_0 + b_i y_0 + \sqrt{a_i^2 + b_i^2} r \le c_i, \ \forall i \\ & r \ge 0, \ x_0, y_0 \in \mathbb{R} \end{aligned}$$

There are 3 decision variables and $O(n)$ constraints.

Problem 4

In this problem, we will prove optimality of Dijkstra's algorithm through complementary slackness conditions of a carefully chosen linear program.

(a) Consider computing a shortest s-t path in a directed graph $G(V, E)$ with non-negative edge costs c_e for each edge $e \in E$. Prove that every simple s-t path of G corresponds to a 0-1 feasible solution of the following linear program with the same object function value:

$$\min_{e} \sum_{e \in E} c_e x_e$$

subject to:

$$\begin{aligned} \sum_{e \in \delta^+(S)} x_e \geq 1 &\quad \text{for all } S \subset V \text{ with } s \in S, t \notin S \\ x_e \geq 0 &\quad \text{for all } e \in E. \end{aligned}$$

Note that $\delta^+(S)$ denotes the edges sticking out of the set S. Formally, $\delta^+(S) = \{(u \to v) \in E : u \in S, v \notin S\}$.

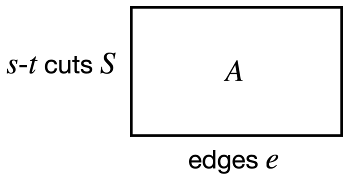
(b) What is the dual of this linear program?

(c) What are the complementary slackness conditions?

(d) Let P be the shortest s-t path returned by Dijkstra's algorithm. Prove that the solution to the linear program in (a) corresponding to P is an optimal solution, by exhibiting a dual feasible solution to the dual program in (b) such that complementary slackness conditions hold. [Hint: it is enough to use only dual variables of the form y_S for subsets $S \subseteq V$ correspond to the first i vertices processed by Dijkstra's algorithm. (for some i)]

Solution:
(a) Notice that a set $S \subset V$ with $s \in S$ an $t \notin S$ defines an s-t cut. Therefore, on any s-t path P, there must be at least one edge, say $e \in P$, that crosses the cut $e \in \delta^+(S)$. It follows that for any s-t path P, setting $x_e = 1$ for all $e \in P$, and zero otherwise gives a primal feasible solution. For such a 0-1 solution, the objective function value equals the total cost of the edges on the path.

(b) Let $\mathcal{S}$ be the set of all s-t cuts of G. We rewrite the LP in matrix vector form: $\min \mathbf{c}^T x$; subject to $Ax \geq \mathbf{1}$ ($\mathbf{c}$ is a vector of edge costs, and $\mathbf{1}$ is a vector of 1's.) The rows of A are indexed by s-t cuts $S \in \mathcal{S}$. The columns of A are indexed by edges E. The entries of A are



$$A_{S,e} = \begin{cases} 1 & \text{if } e \in \delta^+(S) \\ 0 & \text{otherwise} \end{cases}.$$

We create a dual variable y_S for each constraint of the primal problem corresponding the s-t cut $S \in \mathcal{S}$. From the standard dual formula: $\max \mathbf{1}^T y$; subject to $A^T y \leq \mathbf{c}$. Now the tricky part, What is A^T? Consider the column of A corresponding to edge e. The rows of this column are 1 if $e \in \delta^+(S)$ and 0 otherwise. This means the row for edge e in $A^T y$ is the sum over all s-t cuts $S \in \mathcal{S}$ where $e \in \delta^+(S)$. Therefore, the dual linear program is:

$$\begin{aligned} & \max \sum_{S \in \mathcal{S}} y_S \\ \text{s.t.} \quad & \sum_{S \in \mathcal{S}:\ e \in \delta^+(S)} y_S \leq c_e, \ \forall e \in E \\ & y_S \geq 0. \end{aligned}$$

(c) Recall that there is one dual variable for each primal constraint, and one dual constraint for each primal variable. For each primal variable x_e (for edge $e \in E$) there is corresponding dual constraint (indexed by the same edge e). For each dual variable y_S (for s-t cut $S \in \mathcal{S}$) there is a corresponding primal constraint (indexed by the same cut S). Complementary slackness conditions require that if the primal variable is not zero, then the corresponding dual constraint holds with equality (and vice-versa). Therefore, we have:

$$\text{if } x_e > 0, \quad \text{then} \sum_{S \in \mathcal{S}:\ e \in \delta^+(S)} y_S = c_e$$
$$\text{if } y_S > 0, \quad \text{then} \sum_{e \in \delta^+(S)} x_e = 1.$$

(d) To simplify the presentation we will assume that all shortest paths are unique. The main steps of the analysis still work without this assumption but there are few extra details.

Following the hint, we track the progress of Dijkstra's algorithm and use its current partial solution at iteration i to assign values to dual variables. Recall that on each iteration, Dijkstra's pops some new vertex, say v, off the priority queue and tries to extend shortest paths from v to its neighbors. Define S_i as the set of all vertices explored by the *end* of iteration i. That is, S_i are the first i vertices found (they are also the i closest vertices but we can't assume that). Note that we define $S_1 = \{s\}$.

We assign dual variables as follows. For each edge e sticking out of S_i, let $slack(e)$ denote the value needed to make the dual constraint corresponding to e tight:

$$slack(e) = c_e - \sum_{S \in \mathcal{S}:\ e \in \delta^+(S)} y_S = c_e - \sum_{j=1}^{i} 1(e \in \delta^+(S_j)) y_{S_j}.$$

The i'th vertex visited by Dijkstra's, say v_i, connects to some vertex $u \in S_{i-1}$ along the edge $e = (u, v_i)$ that sticks out of S_{i-1}. We set $y_{S_{i-1}} = slack(u, v_i)$. As a result, the dual constraint for edge e becomes tight.

If we can show dual feasibility, then our choice of dual values implies that complementary slackness holds, and it follows that Dijkstra's algorithm found the shortest s-t path. For each edge e_j of the shortest s-t path, we set $x_{e_j} = 1$ and all other $x_e = 0$. By part (a) any 0-1 solution given by an s-t path is primal feasible. If shortest paths are unique, then exactly one edge of the s-t path sticks out of any of the sets S_i. Therefore, for any $y_{S_i} > 0$, the corresponding primal constraint is tight. Finally, for each edge e used to reach *any* new vertex in the run of Dijkstra's algorithm we ensure the corresponding dual constraint of e is tight. It remains to verify dual feasibility.

Claim: Let u_i be the last vertex added to S_i (the last vertex explored by Dijkstra's algorithm). At the end of iteration i, for any edge $e = (v, w)$ sticking out of S_i where $v \in S_i$ and $w \notin S_i$, we have

$$\mathit{slack}(v, w) = \mathit{dist}(v) + c(v, w) - \mathit{dist}(u_i), \tag{1}$$

where $\mathit{dist}(v)$ is the s to v distance estimate from Dijkstra's algorithm.

First, observe that the term $\mathit{dist}(v) + c(v, w)$ of (1) is the distance to reach vertex w from v on the edge (v, w). These values determine the priority Dijkstra's algorithm visits the vertices. Therefore, if the Claim holds, then Dijkstra's algorithm always picks a vertex v_i that connects to S_{i-1} on an edge e_i with minimum $\mathit{slack}(e_i)$ since this corresponds to the unexplored vertex with minimum estimated distance from s. Updating dual variables by choosing by the minimum slack ensures that the slack on all edges sticking out of S_i are non-negative. From the definition of slack, this implies no dual constraints are violated.

Proof (Claim): We proceed by induction. In the base case $i = 1$, we have $S_1 = \{s\}$. For each neighbor v of s, $\mathit{slack}(s, v) = c(s, v) = \mathit{dist}(s) + c(s, v) - \mathit{dist}(s)$. Now suppose (1) holds after iteration $i - 1$ for all edges (v, w) sticking out of S_{i-1}. In iteration i, Dijkstra's algorithm explores a vertex u_i that was reached from vertex $z \in S_{i-1}$ along edge (z, u_i) with minimum slack. At this point Dijkstra's algorithm finds that $\mathit{dist}(u_i) = \mathit{dist}(z) + c(z, u_i)$. We set the dual variable $y_{S_{i-1}} = \mathit{slack}(z, u_i)$. Consider, any edge (v, w) sticking out of S_i with $v \in S_i$ and $w \notin S_i$. After iteration i, we have that

$$\begin{aligned}
\mathit{slack}(v, w) = c(v, w) - \sum_{j=1}^{i-1} y_{S_j} &= \underbrace{c(v, w) - \sum_{j=1}^{i-2} y_{S_j}}_{\text{slack at the end of iteration } i-1} - y_{S_{i-1}} \\
&\overset{(a)}{=} \mathit{dist}(v) + c(v, w) - \mathit{dist}(u_{i-1}) - y_{S_{i-1}} \\
&\overset{(b)}{=} \mathit{dist}(v) + c(v, w) - \mathit{dist}(u_{i-1}) - \big(\mathit{dist}(z) + c(z, u_i) - \mathit{dist}(u_{i-1})\big) \\
&= \mathit{dist}(v) + c(v, w) - \big(\mathit{dist}(z) + c(z, u_i)\big) \\
&\overset{(c)}{=} \mathit{dist}(v) + c(v, w) - \mathit{dist}(u_i).
\end{aligned}$$

In the above, (a) is by the induction hypothesis, (b) uses the induction hypothesis and the definition of $y_{S_{i-1}}$, and (c) uses Dijkstra's updated distance estimate for u_i.