CS 170 HW 10

Due **2019-04-13, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write none.

2 Opting for releasing your solutions

We are considering releasing a subset of homework submissions written by students for students to see what a full score submission looks like. If your homework solutions are well written, we may consider releasing your solution. If you wish that your solutions not be released, please respond to this question with a "No, do not release any submission to any problems". Otherwise, say "Yes, you may release any of my submissions to any problems".

3 Zero-Sum Games

Alice and Bob are playing a zero-sum game whose payoff matrix is shown below. The ij^{th} entry of the matrix shows the payoff that Alice receives if she plays strategy i and Bob plays strategy j. Alice is the row player and is trying to maximize her payoff, and Bob is the column player trying to minimize Alice's payoff.

Alice \Bob	1	2
1	4	1
2	2	5

Now we will write a linear program to find a strategy that maximizes Alice's payoff. Let the variables of the linear program be x_1, x_2 and p, where x_i is the probability that Alice plays strategy i and p denotes Alice's payoff.

(a) Write the linear program for maximizing Alice's payoff. (Hint : You should think of setting up the constraints of the program such that it finds the best worst case strategy. This would depend on the strategy Bob plays assuming he knows what Alice's (probabilistic) strategy is.)

(b) Eliminate x_2 from the linear program and write it in terms of p and x_1 alone.

(c) Draw the feasible region of the above linear program in p and x_1. You are encouraged to use a plotting software for this.

(d) Write a linear program from Bob's perspective trying to minimizing Alice's payoff. Let the variables of the linear program be y_1, y_2 and p, where y_i is the probability that Bob plays strategy i and p denotes Alice's payoff.

(e) What is the optimal solution and what is the value of the game?

Solution:

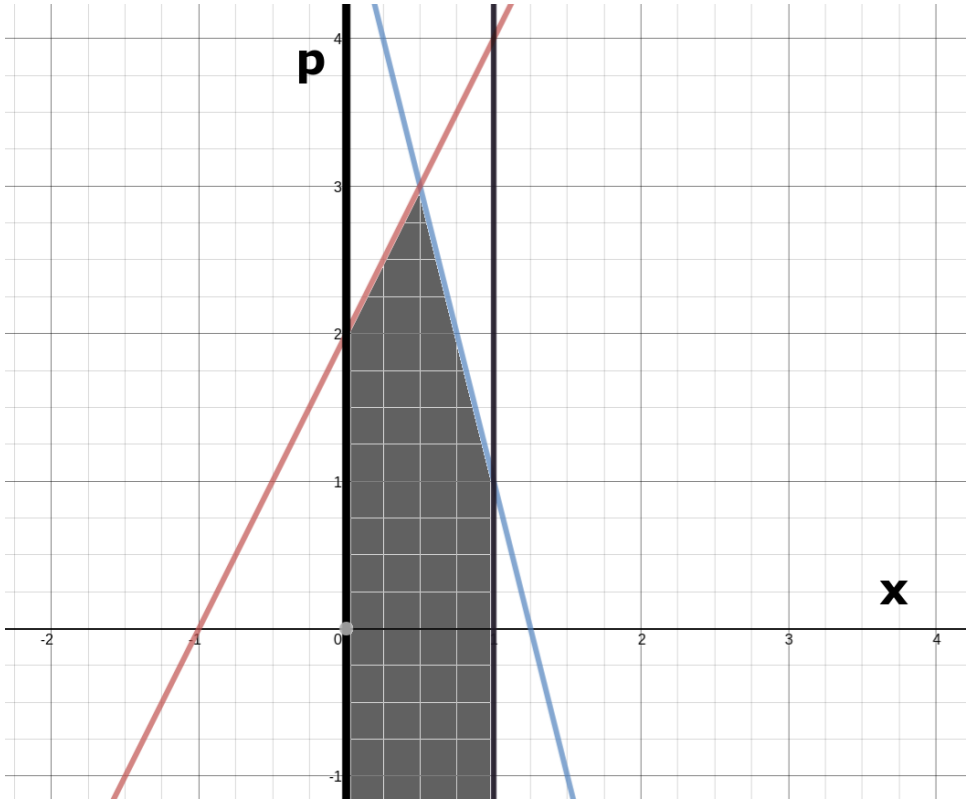
(a)

$$\begin{aligned} \max\, & p \\ & p \leq 4x_1 + 2x_2 \\ & p \leq x_1 + 5x_2 \\ & x_i \geq 0 \\ & x_1 + x_2 = 1 \end{aligned}$$

(b)

$$\begin{aligned} \max\ & p \\ & p \leq 2x_1 + 2 \\ & p \leq -4x_1 + 5 \\ & x_1 \geq 0 \\ & x_1 \leq 1 \end{aligned}$$

(c)



(d)

$$\begin{aligned} \min\, & p \\ p &\geq 4y_1 + y_2 \\ p &\geq 2y_1 + 5y_2 \\ y_i &\geq 0 \\ y_1 + y_2 &= 1 \end{aligned}$$

(e)

$$\begin{aligned} x_1 &= \frac{1}{2} \\ x_2 &= \frac{1}{2} \\ p &= 3 \end{aligned}$$

4 Zero-Sum Battle

Two Pokemon trainers are about to engage in battle! Each trainer has 3 Pokemon, each of a single, unique type. They each must choose which Pokemon to send out first. Of course each trainer's advantage in battle depends not only on their own Pokemon, but on which Pokemon their opponent sends out.

The table below indicates the competitive advantage (payoff) Trainer A would gain (and Trainer B would lose). For example, if Trainer B chooses the fire Pokemon and Trainer A chooses the rock Pokemon, Trainer A would have payoff 2.

		Trainer B: ice	water	fire
Trainer A:	dragon	-10	3	3
	steel	4	-1	-3
	rock	6	-9	2

Feel free to use an online LP solver to solve your LPs in this problem.
Here is an example of an online solver that you can use: `https://online-optimizer.appspot.com/`.

1. Write an LP to find the optimal strategy for Trainer A. What is the optimal strategy and expected payoff?

2. Now do the same for Trainer B. What is the optimal strategy and expected payoff?

Solution:

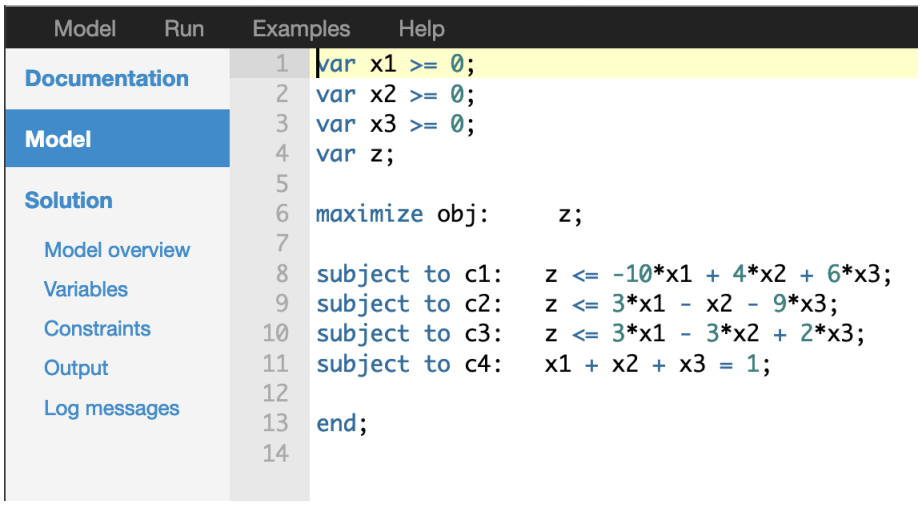
1. d = probability that A picks the dragon type
   s = probability that A picks the steel type

r = probability that A picks the rock type

$$\begin{aligned} \max \quad & z & \\ -10d + 4s + 6r &\geq z & \text{(B chooses ice)} \\ 3d - s - 9r &\geq z & \text{(B chooses water)} \\ 3d - 3s + 2r &\geq z & \text{(B chooses fire)} \\ d + s + r &= 1 & \\ d, s, r &\geq 0 & \end{aligned}$$

The optimal strategy is $d = 0.3346$, $s = 0.5630$, $r = 0.1024$ for an optimal payoff of -0.48.

If you are using the website suggested in this problem, here is what you should put in the model tab:

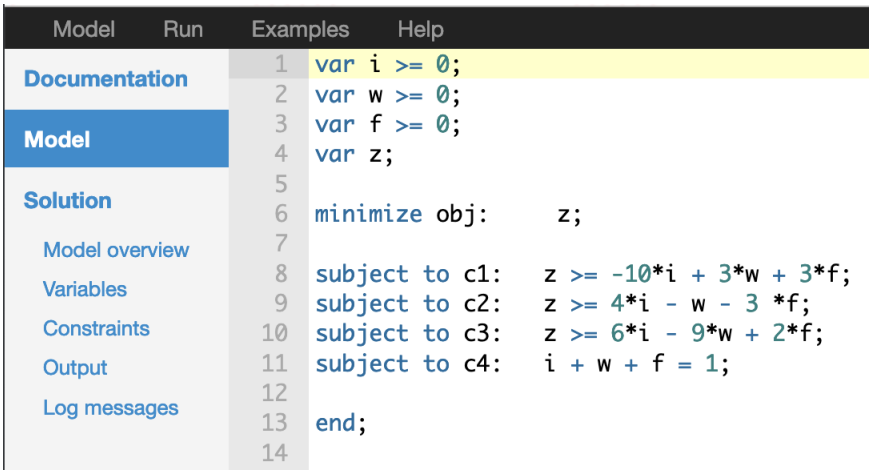


2. i = probability that B picks the ice type
w = probability that B picks the water type
f = probability that B picks the fire type

$$\begin{aligned} \min \quad & z & \\ -10i + 3w + 3f &\leq z & \text{(A chooses dragon)} \\ 4i - w - 3f &\leq z & \text{(A chooses steel)} \\ 6i - 9w + 2f &\leq z & \text{(A chooses rock)} \\ i + w + f &= 1 & \\ i, w, f &\geq 0 & \end{aligned}$$

B's optimal strategy is $i = 0.2677$, $w = 0.3228$, $f = 0.4094$. The value for this is -0.48, which is the payoff for A. The payoff for B is 0.48, since the game is zero-sum.

If you are using the website suggested in this problem, here is what you should put in the model tab:



(Note for grading: Equivalent LPs are of course fine. It is fine for part (b) to maximize B's payoff instead of minimizing A's. For the strategies, fractions or decimals close to the solutions are fine, as long as the LP is correct.)

5 How to Gamble With Little Regret

Suppose that you are gambling at a casino. Every day you play at a slot machine, and your goal is to minimize your losses. We model this as the experts problem. Every day you must take the advice of one of n experts (i.e. a slot machine). At the end of each day t, if you take advice from expert i, the advice costs you some c_i^t in $[0, 1]$. You want to minimize the regret R, defined as:

$$R = \frac{1}{T}\left(\sum_{t=1}^{T} c_{i(t)}^t - \min_{1\le i\le n} \sum_{t=1}^{T} c_i^t\right)$$

where $i(t)$ is the expert you choose on day t. Your strategy will be probabilities where p_i^t denotes the probability with which you choose expert i on day t. Assume an all powerful adversary (i.e. the casino) can look at your strategy ahead of time and decide the costs associated with each expert on each day. Let C denote the set of costs for all experts and all days. Compute $\max_C(\mathbb{E}[R])$, or the maximum possible (expected) regret that the adversary can guarantee, for each of the following strategies.

(a) Choose expert 1 at every step, i.e. $p_1^t = 1$ for all t.

(b) Any deterministic strategy, i.e. for each t, there exists some i such that $p_i^t = 1$.

(c) Always choose an expert according to some fixed probability distribution at every time step. That is, fix some $p_1, \ldots, p_n$, and for all t, set $p_i^t = p_i$.

What distribution minimizes the regret of this strategy? In other words, what is $\text{argmin}_{p_1,\ldots,p_n} \max_C(\mathbb{E}[R])$?

This analysis should conclude that a good strategy for the problem must necessarily be randomized and adaptive.

Solution:

(a) 1. Consider the case where the cost of expert 1 is always 1 and the costs of all the other experts are always 0. Thus $\sum_{t=1}^{T} c_{i(t)}^t = \sum_{t=1}^{T} c_1^t = T$, and $\min_{1\le i\le n} \sum_{t=1}^{T} c_i^t = 0$, so the regret is $R = \frac{1}{T}(T-0) = 1$.

(b) $\frac{n-1}{n}$. Consider the case where the cost of the chosen expert is always 1, and the cost of each other expert is 0. Let k be the least-frequently chosen expert, and let m_k be the number of times that expert is chosen. This will result in a regret of $\frac{1}{T}(T-m_k)$

Since the best expert is the one that is chosen least often, the best strategy will try to maximize the number of times we choose the expert that is chosen least often. This means we want to choose all the experts equally many times, so expert k is chosen in at most T/n of the rounds. Therefore, $m_k \le \frac{T}{n}$, thus the regret is at least $\frac{1}{T}(T-\frac{T}{n}) = \frac{n-1}{n}$.

(c) $(1-\min_i p_i)$. Like in part (a), the distribution is fixed across all days, so we know ahead of time which expert will be chosen least often in expectation. Let $k = \text{argmin}_i p_i$ be the expert with least cost. Let $c_k^t = 0$ for all t, and let $c_i^t = 1$ for all $i \neq k$ and for all t. This way, $\mathbb{E}\left[\sum_{t=1}^{T} c_{i(t)}^t\right] = T\left(\sum_{i\neq k} p_i \cdot 1 + p_k \cdot 0\right) = T(1-p_k)$. We also have $\min_i \sum_{t=1}^{T} c_i^t = 0$, so we end up with an expected regret of $\frac{1}{T}(T(1-p_k)-0) = 1-p_k$.

To minimize the expectation of R is the same as maximizing $\min_i p_i$, which is achieved by the uniform distribution. This gives us regret $\frac{n-1}{n}$ (this is the same worst case regret as in part (b)).

6 Solving Linear Programs using Multiplicative Weights

In this problem, we will develop an algorithm to approximately solve linear programs using multiplicative weights. For simplicity, we will restrict our attention to a specific kind of linear programs given as follows:

Given vectors $\mathbf{a}_j$ for $j = 1, \ldots, m$ such that $|(a_j)_i| \leq 1$ where $i = 1, \ldots, n$, the linear program on variables $\mathbf{x} = (x_1, \ldots, x_n)$ is given by:

$$\begin{aligned} & & & \max(0) \\ \text{for all} \quad & j = 1, \ldots, m : & & \langle \mathbf{a}_j, \mathbf{x} \rangle \leq 0 \\ & & & \sum_{i=1}^{n} x_i = 1 \\ \text{for all} \quad & i = 1, \ldots, n : & & x_i \geq 0 \end{aligned}$$

Here $\langle \mathbf{x}, \mathbf{y} \rangle$ denotes the dot product between vectors, specifically $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{n} x_i \cdot y_i$ where x_i, y_i refers to the i^{th} components of the vector $\mathbf{x}, \mathbf{y}$.

We will now use multiplicative weights update towards solving our linear program. The idea would be to execute multiplicative weights update against an appropriate adversary (sequence of losses) and let the weights of the algorithm determine the solution $\mathbf{x}$. Formally, the rough outline of our algorithm to solve the above LP is as follows:

For $t = 1$ to a sufficiently large number T

1. Multiplicative weights update will return a current probability distribution $\mathbf{x}^{(t)}$
2. If $\mathbf{x}^{(t)}$ approximately satisfies the LP, return $\mathbf{x}^{(t)}$.
3. Adversary $\mathcal{A}$ will present a loss vector $\ell^{(t)} \in \mathbb{R}^n$.
4. Multiplicative weights algorithm will incur a loss of $\langle \ell^{(t)}, \mathbf{x}^{(t)} \rangle$ and update its weights.

Return $\mathbf{x}^{(T)}$

We will design an algorithm that will act as the adversary $\mathcal{A}$, so that $\mathbf{x}^{(T)}$ is an approximate solution to the LP.

Recall that the multiplicative weights algorithm obeys the following regret bound:

Theorem. The multiplicative weights algorithm starts with an iterate $\mathbf{x}^{(1)} \in \mathbb{R}^n$, and then suffers a sequence of loss vectors $\ell^{(1)}, \ldots, \ell^{(T)} \in \mathbb{R}^n$ such that $\ell_i^{(t)} \in [-1, 1]$. It then produces a sequence of iterates $\mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(T)}$ such that for some $0 < \epsilon \leq \frac{1}{2}$,

$$\sum_{t=1}^{T} \langle \ell^{(t)}, \mathbf{x}^{(t)} \rangle \leq \min_{1 \leq i \leq n} \left(\sum_{t=1}^{T} \ell_i^{(t)} \right) + 2\left(\epsilon T + \frac{\log(n)}{\epsilon} \right)$$

(a) Let $\mathbf{p} = (p_1, \ldots, p_n)$ be a probability distribution over $\{1, \ldots, n\}$, i.e., $p_i \geq 0$ and $\sum_{i=1}^n p_i = 1$. For any vector $\mathbf{v} \in \mathbb{R}^n$ prove that,

$$\min_{i=1}^{n} v_i \leq \sum_{i=1}^{n} p_i v_i$$

(In words, the minimum is smaller than the average under every distribution)

(b) We will now observe an important property of multiplicative weights. The algorithm not only has small regret against any fixed best expert, but also has small regret against any fixed probability distribution over experts.

Let $\mathbf{p}^* = (p_1^*, \ldots, p_n^*)$ be a probability distribution on $\{1, \ldots, n\}$, i.e. $p_i^* \geq 0$ for each $1 \leq i \leq n$ and $\sum_{i=1}^n p_i^* = 1$. Using the bounds in the regret of the multiplicative weights algorithm mentioned above, prove that

$$\sum_{t=1}^{T} \Big(\langle \ell^{(t)}, \mathbf{x}^{(t)} \rangle - \langle \ell^{(t)}, \mathbf{p}^* \rangle \Big) \leq 2 \Big(\frac{\log(n)}{\epsilon} + \epsilon T \Big)$$

(c) At the t^{th} iteration, the probability distribution $\mathbf{x}^{(t)}$ output by multiplicative weights update is a *candidate* solution to our LP. But it is likely to violate some of the constraints of the LP, namely $\langle \mathbf{a}_i, \mathbf{x}^{(t)} \rangle \leq 0$.

Describe how to implement the adversary $\mathcal{A}$ to nudge the multiplicative weights algorithm towards a feasible solution to the LP.

Hint: What loss vector should the adversary $\mathcal{A}$ pick in order to penalize the multiplicative weights algorithm the most for violations?

(d) Let us call a solution $\mathbf{x}$ to be η-approximate solution to the LP if it satisfies all the constraints within an error of η where $\eta \leq 2$, i.e.,

$$\langle \mathbf{a}_i, \mathbf{x} \rangle \leq \eta$$

.

Assume that the LP is feasible in that there exists some probability distribution $\mathbf{x}^*$ that satisfies all the constraints. Show that if multiplicative weights uses $\epsilon = \frac{\eta}{4}$ then after $T = \frac{16 \log(n)}{\eta^2}$ iterations, the distribution $\mathbf{x}^{(T)}$ is an η-approximate solution.

Hint: In every iteration t, if the current distribution $\mathbf{x}^{(t)}$ violates a constraint by more than η, then it accumlates *regret* for not being the feasible solution $\mathbf{x}^*$

Solution:

(a) $\sum_{i=1}^n p_i v_i \geq \sum_{i=1}^n p_i \cdot (\min_{i=1}^n v_i) \geq (\min_{i=1}^n v_i) \cdot 1$

(b) Recall the regret bound for multiplicative weights:

$$\sum_{t=1}^{T}\langle \ell^{(t)}, \mathbf{x}^{(t)}\rangle \leq \min_{1\leq i\leq n}\left(\sum_{t=1}^{T}\ell_i^{(t)}\right) + 2\Big(\frac{\log(n)}{\epsilon}+\epsilon T\Big).$$

Use the claim from the previous subpart of the problem, with the vector $\mathbf{v} = (v_1, \ldots, v_n)$, $v_i = \sum_{t=1}^{T}\ell_i^{(t)}$.

$$\min_{1\leq i\leq n}\left(\sum_{t=1}^{T}\ell_i^{(t)}\right) \leq \sum_{i=1}^{n} p_i^*\left(\sum_{t=1}^{T}\ell_i^{(t)}\right) = \sum_{t=1}^{T}\langle \ell^{(t)}, p^*\rangle$$

Putting the two inequalities together, we get that

$$\sum_{t=1}^{T}\langle \ell^{(t)}, \mathbf{x}^{(t)}\rangle \leq \sum_{t=1}^{T}\langle \ell^{(t)}, p^*\rangle + 2\Big(\frac{\log(n)}{\epsilon}+\epsilon T\Big)$$

Rearranging the terms we get

$$\sum_{t=1}^{T}\Big(\langle \ell^{(t)}, \mathbf{x}^{(t)}\rangle - \langle \ell^{(t)}, p^*\rangle\Big) \leq 2\Big(\frac{\log(n)}{\epsilon}+\epsilon T\Big)$$

(c) In any iteration t, the adversary $\mathcal{A}$ picks the most violated constraint $\mathbf{a}_j$ as the loss vector. Specifically,

$$\ell^{(t)} = \mathbf{a}_{j^*}$$

where $j^* = \text{argmax}_{j=1}^{m}\langle \mathbf{a}_j, \mathbf{x}^{(t)}\rangle$

Notice that the above adversary can be easily implemented by just scanning the constraints and finding the one that is most violated.

(d) From the bound $|(\mathbf{a}_j)_i| \leq 1$ on the coefficients in our LP, we get that the losses induced by the adversary $\mathcal{A}$ are bounded in $[-1, 1]$.
We will proceed to prove this question via contradiction, assuming that at day T the multiplicative weight algorithm has not produced an η-approximate solution.
If $\mathbf{x}^{(t)}$ is not an η-approximate solution, there must exist an i such that,

$$\langle \mathbf{a}_i, \mathbf{x}^{(t)}\rangle \geq \eta$$

and the adversary $\mathcal{A}$ returns the loss $\ell^{(t)} = \mathbf{a}_i$ as from part c the loss vector is the largest a_i.
Thus in this round, the loss of the MW algorithm is $\langle \mathbf{a}_i, \mathbf{x}^{(t)}\rangle \geq \eta$. A feasible solution x^* has a loss of $\langle \ell^{(t)}, \mathbf{x}^*\rangle = \langle \mathbf{a}_i, \mathbf{x}^*\rangle \leq 0$. Therefore, the regret against of MW algorithm against x^* is at least $\eta - 0 = \eta$.
So we will now apply the regret bound from the subpart (b) with $\mathbf{p}^* = \mathbf{x}^*$.

$$\sum_{t=1}^{T}\Big(\langle \ell^{(t)}, \mathbf{x}^{(t)} - \langle \ell^{(t)}, \mathbf{x}^*\rangle\rangle\Big) \leq 2\Big(\frac{\log(n)}{\epsilon}+\epsilon T\Big)$$

Since the regret in each round is at least η, we get that

$$\eta T \leq 2\Big(\frac{\log(n)}{\epsilon} + \epsilon T\Big)$$

Setting $\epsilon = \frac{\eta}{4}$ and $T = \frac{16\log(n)}{\eta^2}$, we get:

$$\eta\frac{16\log(n)}{\eta^2} \leq 2\Big(\frac{\log(n)}{\frac{\eta}{4}} + \frac{\eta}{4}\cdot\frac{16\log(n)}{\eta^2}\Big)$$

$$\frac{16\log(n)}{\eta} \leq 2\Big(\frac{4\log(n)}{\eta} + \frac{4\log(n)}{\eta}\Big)$$

$$\frac{16\log(n)}{\eta} \leq 2\Big(\frac{8\log(n)}{\eta}\Big)$$

$$\frac{16\log(n)}{\eta} \leq \frac{16\log(n)}{\eta}$$

i.e., it cannot be that the MW algorithm has a regret of η for $T > \frac{16\log(n)}{\eta^2}$ steps so it indeed produces an η-approximate solution to the LP after T iterations.

7 Restoring the Balance!

We are given a network $G = (V, E)$ whose edges have integer capacities c_e, and a maximum flow f from node s to node t. Explicitly, f is given to us in the representation of integer flows along every edge e, (f_e).
However, we find out that one of the capacity values of G was wrong: for edge (u, v), we used c_{uv} whereas it really should have been $c_{uv} - 1$. This is unfortunate because the flow f uses that particular edge at full capacity: $f_{uv} = c_{uv}$. We could run Ford Fulkerson from scratch, but there's a faster way.

Describe an algorithm to fix the max-flow for this network in $O(|V| + |E|)$ time. Also give a proof of correctness and runtime justification.

Solution:

Main Idea: Since we know the flows along every edge, we can construct the residual graph in $O(|V| + |E|)$ as there are $2|E|$ edges in the residual graph. The original edges from G in the residual will have weight $c_{uv} - f_{uv}$ and the back edges will have weight f_{uv}.

In this residual graph with the original capacity for (u, v), use DFS to find a path from t to v and from u to s. Join these paths by adding the edge (v, u) in between them to get a path p. Update f by pushing 1 unit of flow from t to s on p (i.e. for each $(i, j) \in p$, reduce f_{ji} by 1, and increase f_{ij} by 1 for all edges except (u, v)). Finally, use DFS to see if the residual graph of the resulting graph, has an $s-t$ path. If so, push 1 unit of flow on this path.

Proof of Correctness: Consider the residual graph of G. If (u, v) is at capacity, then there is a flow of at least 1 unit going from v to t, and since f_e are integral there is a path from v to t with at least one unit of flow moving through it. So the residual graph will have

a path from t to v. Similarly, there will be a path from u to s in the residual graph, so the algorithm can always find p correctly.

Note that the size of the maximum flow in the new network will be either the same or 1 less than the previous maximum flow (because changing one edge can change the capacity of the min-cut by at most 1). In the former case, after pushing flow from t to s on p, it is possible to push more flow from s to t, so there must be an $s - t$ path in the new residual graph, so the Ford Fulkerson algorithm will find and push 1 unit of flow because all capacities and flow values are integral. Thus the algorithm finds the new max flow correctly. In the latter case, we will already have the max flow after pushing flow backwards on p, so our algorithm is still correct.

Runtime: The runtime is $O(|V| + |E|)$ because the algorithm just calls DFS on the residual thrice, and it takes $O(|V| + |E|)$ to construct the residual.

Note: for grading, it must be explained how you get the residual graph as it is not given you in the problem statement.