COS 445 - PSet 3

Due online Monday, March 25th at 11:59 pm.

Instructions:

- Some problems will be marked as *no collaboration* problems. This is to make sure you have experience solving a problem start-to-finish by yourself in preparation for the midterms/final. You cannot collaborate with other students or the Internet for these problems (you may still use the referenced sources and lecture notes). You may ask the course staff clarifying questions, but we will generally not give hints.
- Submit your solution to each problem as a **separate PDF** to codePost. Please make sure you're uploading the correct PDFs to the correct locations![1] If you collaborated with other students, or consulted an outside resource, submit a (very brief) collaboration statement as well. Please anonymize your submission, although there are no repercussions if you forget.
- The cheatsheet gives problem solving tips, and tips for a "good proof" or "partial progress."
- Please reference the course collaboration policy here.

For convenience, we restate some definitions used in this problem set.

[1] We will assign a minor deduction if we need to maneuver around the wrong PDFs. Please also note that depending on if/how you use Overleaf, you may need to recompile your solutions in between downloads to get the right files.

Problem 1: Linear Programming (20 points, no collaboration)

Alice is trying to get enough oranges and bananas to host a fruit party. To successfully host a party she needs **at least 9 oranges** and **at least 5 bananas**. Unfortunately, her local grocery story only sells fruit in bundles. Bundle A costs 8 dollars and contains 7 oranges and 3 bananas. Bundle B costs 9 dollars and contains 5 oranges and 7 bananas. Fortunately, the grocery story will allow Alice to buy fractions of bundles (i.e. she can buy 2.5 bundle As). They will not allow Alice to buy negative bundles (i.e. she cannot buy -1 bundle As and 3 bundle Bs).

Alice would like to buy x_A bundle As and x_B bundle Bs to guarantee she has at least 9 oranges and at least 5 bananas. Moreover, she would like to find the solution that minimizes her dollars spent.

Part a (10 points)

Write a linear program whose solution is the optimal choice of x_A, x_B for Alice's problem.

Part b (10 points)

Take the dual of the linear program from part a.

Problem 2: An Algorithm for Nash (40 points)

A *feasibility* LP is an LP but without the objective function. That is, a feasibility LP is just a list of inequalities (which might be written as $\leq, \geq$ or $=$). A feasibility LP either outputs some $\vec{x}$ that satisfies all the inequalities (if a solution exists), or outputs "no" if no such solution exists.

Part 0: Examples

Here are three examples of feasibility LPs, if you would find it helpful. If you feel that you understand the definition, you should skip this section.

- Variables: x, y.
- Constraints:
  1. $4x \leq 2y$.
  2. $2x + 8y \geq 4$.
  3. $x + y = 0$.

One solution to this feasibility LP is $x = -10, y = 10$. Another solution is $x = -5, y = 5$. Another is $x = -1, y = 1$. Here is another example of a feasibility LP:

- Variables: x, y.
- Constraints:
  1. $4x \geq 2y$.
  2. $2x + 8y \geq 4$.
  3. $x + y = 0$.

The solution to this feasibility LP is "infeasible." That is, there is no (x, y) satisfying all three equations.[2] Here is one last example of a feasibility LP:

- Variables: x_i, for $i \in [n]$.
- Constraints:
  1. $x_i \geq 0$, for all $i \in [n]$.
  2. $\sum_{i=1}^{n} j \cdot 2^i \cdot x_i \leq 2^j$, for all $j \in [n]$.

One solution to this feasibility LP is $x_i = 0$ for all i. Another solution is $x_1 = 1$, $x_i = 0$ for all $i > 1$.

[2] To see this, add the first two equations together to get $6x + 6y \geq 4$. Then, add -6 times the third equation to get $0 \geq 4$, which is a contradiction. Observe that if any x, y existed which satisfied all three equations, then such an x, y would necessarily satisfy the sum of the first two plus -6 times the last one. But this equation is unsatisfiable, because it is false.

Part a (25 points)

Consider a two-player (not-necessarily-zero-sum) game, where each player has n pure actions labeled $\{1, \ldots, n\}$. The payoff to player i when player 1 uses pure action x and player 2 uses pure action y is $p_i(x, y)$.

Fix a subset $S_1 \subseteq [n]$ of pure actions for player 1, and a subset $S_2 \subseteq [n]$ of pure actions for player 2. Write a feasibility LP to determine whether there exists a mixed strategy $\vec{z}$ for player one, and $\vec{w}$ for player two, for this game which satisfies all four of the following properties (and prove that your feasibility LP is correct, even if your proof is brief):

1. Every pure action in S_1 is a best response for player 1 to $\vec{w}$ (pure actions not in S_1 may or may not be best responses).
2. In $\vec{z}$, player 1 only uses pure actions in S_1, and does not use pure actions $\notin S_1$.
3. Every pure action in S_2 is a best response for player 2 to $\vec{z}$ (pure actions not in S_2 may or may not be best responses).
4. In $\vec{w}$, player 2 only uses pure actions in S_2, and does not use pure actions $\notin S_2$.

If it helps to be explicit, you are given as input: $p_i(x, y)$ for both players $i \in \{1, 2\}$ and all $x, y \in [n]$, and $S_1, S_2 \subseteq [n]$. These are fixed, and not variables. Your goal is to produce a feasibility LP, which will output an $(\vec{z}, \vec{w})$ satisfying the above constraints if one exists, or output "infeasible" if it doesn't.

Part b (15 points)

Design an exponential time (in n) algorithm to find a Nash equilibrium in two-player games with n actions. You may use without proof the fact that linear programs can be solved in polynomial time. You may also use without proof the fact that Nash equilibria always exist.

Hint: Your proof should *somewhere* need to assume that Nash equilibria always exist, as otherwise you would be proving Nash's theorem!

Hint: If your solution to part b makes use of your solution to part a, your proof should also *somewhere* need to actually use properties of the particular LP you wrote in part a. Otherwise, your same proof would "work" even if the LP in part a were always infeasible!

Problem 3: Biased Information Cascades (60 points)

I have an urn. Into the urn I put one red ball and one blue ball. I put a third ball in that is red with probability p and blue with probability $1 - p$. One at a time, participants draw a ball from the urn, see its color, and guess whether the urn has more red balls or blue balls (and then put it back). Recall that partipants see their own draw, and all previous *guesses* (but *not* previous draws). Participants are fully rational, trying to guess correctly, and know that every other participant is fully rational. Note that when $p = 1/2$, this is exactly the model from Lecture 11.

Recall the following definition from Lecture 11:

Definition 1 (Information Cascade)**.** *Say that player i's draw is* revealed *if, based on the information available to player i, they would guess the color of their draw (i.e. if it were red, they'd guess red. If it were blue, they'd guess blue). Say that player i's draw is* ignored *if, based on the information available to player i, they would make the same guess no matter their draw (i.e. they would guess red no matter what).*

An information cascade *is when there exists a player i such that all draws $\geq i$ are ignored.*

Hint: For all parts, if you are stuck, you may want to revisit Lecture 11 for the case where $p = 1/2$ and follow that outline. But you should also be aware that while the high-level outline is similar, "the math" may look different.

Note: It is OK if you make an off-by-one error in the definition of when a cascade starts. Please don't stress about this aspect.

Part a (10 points)

Define $C_p(t)$ to be the probability that an information cascade occurs *by time t* when the bias is p (that is, $C_p(t)$ is the probability that player t ignores their draw).

When $p > 2/3$, what is $C_p(t)$?

Note: You should provide an answer for all t. For this range of p, the answer does not depend on p.

Part b (10 points)

When $p > 2/3$, conditioned that a cascade occurs, what is the probability that the cascade is red (note this is *not* asking the probability the cascade is correct)?

Note: The answer in this range is just a number, independent of p.

Part c (20 points)

When $p \in (1/2, 2/3)$, what is $C_p(t)$?

Note: You should provide an answer for all t and all $p \in (1/2, 2/3)$. The answer has a simple form when t is odd, and a different simple form when t is even.

Part d (20 points)

When $p \in (1/2, 2/3)$, conditioned that a cascade occurs, what is the probability that the cascade is red (note this is *not* asking the probability the cascade is correct)?

Note: The answer depends on p, but it is a simple function of p.

Extra Credit: Another Algorithm for Nash

Recall that extra credit is not directly added to your PSet scores, but will contribute to your participation. Some extra credits are **quite** challenging. We do not suggest attempting the extra credit problems for the sake of your grade, but only to engage deeper with the course material. If you are interested in pursuing an IW/thesis in CS theory, the extra credits will give you a taste of what that might be like.[3]

For this problem, you *may* collaborate with any students. You *may not* consult course resources or external resources. In this problem we will guide you through the proof of a well-known result, so you should *not* copy the proof from one of the course texts (nor should you try to find a proof from external sources). You *must* follow the guide below (and not provide an alternative proof).

Consider any *symmetric* two-player game. That is, $p_1(x, y) = p_2(y, x)$ for all x, y. Consider also the following system of inequalities. We'll refer to this as the *Lemke-Howson Polytope*.

- Variables $x_1, \ldots, x_n$.
- (Non-negativity) $x_i \geq 0$ for all i.
- (Responsiveness) $\sum_j x_j p_1(i, j) \leq 1$.

Part a

Say that an action i is *covered in* $\vec{x}$ if either the non-negativity constraint is tight (i.e. $x_i = 0$) or the Responsiveness constraint is tight (i.e. $\sum_j x_j p_1(i, j) = 1$), or both. Prove that if $\vec{x}$ is inside the Lemke-Howson polytope, and $\vec{x} \neq 0$, and all actions i are covered in $\vec{x}$, then $\vec{x}/|\vec{x}|_1$ is a symmetric Nash equilibrium (that is, $\vec{x}/|\vec{x}|_1$ is a best response to itself).

Part b

For this part, you should assume that any set of n equations taken above have a solution, and that this solution is unique.

The Lemke-Howson algorithm starts from the point $\vec{0}$ and repeatedly *pivots*. That is, the current point will always have exactly n tight constraints. The pivot will pick one of these constraints and "relax" it (keeping the other $n - 1$ tight). A new constraint will become tight, and this will be the new point (you do not need to prove that this procedure is well-defined).

From $\vec{0}$, the pivot rule simply picks an arbitrary tight constraint to relax (let's say $x_1 = 0$). This causes a new constraint to become tight. If it's the 1st Responsiveness constraint, then by Part a we've found a Nash and are done! If not, then we have exactly one *double-covered* action. That is, there is some action i such that the non-negativity and Responsiveness constraints are both tight. We pick the non-negativity constraint for i to relax next.

In general, for our current point $\vec{x} \neq \vec{0}$, if there is no double-covered action we terminate (and hope that it's a Nash and not back at $\vec{0}$). If there's a double-covered action, it's because we just made one of the constraints for i tight. So relax the other one and continue.

Prove that the Lemke-Howson algorithm will never revisit a vertex $\vec{y}$ without first revisiting the origin. You may use without proof the fact that if $\vec{z}$ pivots to $\vec{w}$ when constraint C is relaxed,

[3] Keep in mind, of course, that you will do an IW/thesis across an entire semester/year, and you are doing the extra credit in a week. Whether or not you make progress on the extra credit in a week is not the important part — it's whether or not you enjoy the process of tackling an extremely open-ended problem with little idea of where to get started.

causing constraint D to become tight, then $\vec{w}$ pivots to $\vec{z}$ when constraint D is relaxed, causing constraint C to become tight.

Part c

Prove that the Lemke-Howson algorithm cannot ever return to $\vec{0}$. Conclude that the Lemke-Howson algorithm finds a Nash after at most $\binom{2n}{n}$ pivots.
Hint: You may want to prove that the algorithm terminates as soon as 1 becomes covered.