CSD 15-784 - Cooperative AI

Homework 1 – Normal-form games (due Oct. 3 11:59pm US Eastern time)

Instructions

Show all your work. You may work alone or discuss with **one** other person, but you must follow the following rules or it will be considered cheating. If you discuss with another person, you **must** explicitly acknowledge that specific person on your writeup. Also, the only way in which you may work with another person is to work on a whiteboard together, and then when you are done discussing, to erase the whiteboard, without taking any notes or other record with you, other than what you remember. (Using the zoom whiteboard is allowed if you want to meet remotely.) You should write up your code and your writeup alone.

(This homework will be updated later with instructions for how to submit.)

1

In this problem, you will code up an algorithm for computing, given a 2-player game in normal form of arbitrary size, all of the following: (1) the best Nash equilibrium, (2) the worst Nash equilibrium, (3) the best correlated equilibrium, (4) the worst correlated equilibrium. We will take "best" to mean maximizing social welfare (sum of expected utilities), and "worst" means minimizing that. In general, there may be more than one optimal equilibrium and it does not matter which one you return; but you should also return the value (social welfare) of that equilibrium, for which there is a unique answer.

You will likely want to use the (mixed integer) linear programming formulations from class.

Please write your algorithms in Python and submit a `.py` file containing your code. We plan to test the code using Python 3.10 in particular. You are allowed to use `cvxpy` with the `GLPK_MI` solver to solve LPs and MIPs. (You can install this by running `pip install cvxopt`.) You are not allowed to use a library like `nashpy` that directly finds Nash equilibria.

Your code should define four functions named

1. `best_Nash`

2. `worst_Nash`

3. `best_correlated_equilibrium`

4. `worst_correlated_equilibrium`

that perform the four tasks specified above, respectively. (Obviously, you may write further functions and define these four functions in terms of these further functions.) Each of these functions should take as input a `numpy` array of shape $(n, m, 2)$ representing an n-by-m game specifying the payoff matrix of the game. For example, the a Prisoner's Dilemma would be defined as follows:

```
import numpy as np
pd = np.array([[[6,6] , [0,10]],
               [[10,0] , [4,4]]])
```

Your `best_Nash` and `worst_Nash` methods should output a single 3-tuple, consisting of

- the social welfare of the equilibrium;
- a `numpy` array with n entries representing the potentially mixed strategy of Player 1;
- a `numpy` array with m entries representing the potentially mixed strategy of Player 2.

For example, in the Prisoner's Dilemma, the output should be `(8, array([0, 1]), array([0, 1]))` for both of these methods.

Your `best_correlated_equilibrium` and `worst_correlated_equilibrium` functions should output a single 2-tuple consisting of

- the social welfare of the equilibrium;
- an n-by-m `numpy` array representing the correlated strategy of the two players.

For example, in the Prisoner's Dilemma, the output should be `(8, array([[0, 0], [0, 1]]))` for both of these methods.

Please include your own test cases in the Python file you submit.

2

Consider the following computational problem:

NEW-NASH. *You are given a 2-player normal-form game G with at least 2 columns. Consider the game G' that results from removing its rightmost column. You are asked to determine whether there exists a Nash equilibrium of G' that is not an equilibrium of G.*

a. Adapt the mixed integer linear program from class (for finding an optimal Nash equilibrium) to solve this problem. (Refer to the rightmost column as c^*.)

Writing it in mathematical notation is fine; emphasize the "new" parts of the program.

b. Prove the problem is NP-complete. Hints: To show membership, it suffices to show that, *given the supports of the new equilibrium*, you could find that new equilibrium and check that it is in fact new. To show hardness, you may assume the following problem is NP-hard:

IN-SUPPORT. *You are given a 2-player normal-form game G and a number $p > 0$. You are guaranteed that either there is no Nash equilibrium of G that puts positive probability on the first row, or that there is a Nash equilibrium of G that puts at least p probability on the first row. You are asked to determine which of these two possibilities is the case for G.*

As always, be careful about the direction in which you do the reduction. That is, you have to show how an algorithm for NEW-NASH would allow you to also solve IN-SUPPORT.

3

Consider the following n-player version of the Prisoner's Dilemma. For each player i, player i's set of pure strategies is $A_i = \{C, D\}$. The payoffs are given by

$$u_i(a_1, ..., a_n) = \mathbb{1}[a_i = D] + \sum_{j \neq i} 2\mathbb{1}[a_j = C]/(n-1).$$

($\mathbb{1}[P]$ evaluates to 1 if P is a true proposition, and to 0 if P is a false proposition.) Intuitively, each player chooses between generating one unit of utility for herself by defecting, and generating two units of utility to be distributed equally across the other players by cooperating. The unique Nash equilibrium of this game is $(D, ..., D)$ for a utility of 1 for each player. Meanwhile, in $(C, ..., C)$, everyone's utility is 2.

Consider the following two programs from class that achieve (C, C) in program equilibrium in the case of $n = 2$: Cooperate with Copies and ϵ-grounded Fair Bot. For each of these programs, for $n > 2$, give a version of the program such that everyone using that program is an equilibrium, and the result of everyone using that program is the outcome $(C, ..., C)$. (Something counts as a version of CwC if it only checks for program equality without doing more sophisticated analysis of the program or simulating it. Something counts as a version of ϵ-grounded Fair Bot if all it does is simulate other programs with some probability, and it terminates in finite time with probability 1. You may assume there is a commonly agreed upon indexing of the players (say, 1, 2, 3) that is given as part of the input to the programs.)