Lab: Prisoner's Dilemma

CSI 3305: Introduction to Computational Thinking

October 24, 2010

1 Introduction

How can rational, selfish actors cooperate for their common good? This is the essential question at the root of many problems in politics and governance. In one sense, the political realm is a "kill or be killed" environment. There are incentives to take advantage of others in order to avoid becoming a victim. At the same time, there are benefits that come with cooperation. How can rational, selfish actors build enough trust in each other to make cooperation possible?

One way of thinking about this problem is to use the Prisoner's Dilemma (PD). You have most likely seen the PD situation play out in police procedural dramas, like CSI or Law and Order. Two people have committed a crime together. They have both been arrested and the detectives are interviewing them in separate interrogation rooms. Each suspect is presented with the same information: If you tell us what happened first, we'll make sure your accomplice gets a heavy sentence and the District Attorney will give you immunity from prosecution. If your accomplice breaks down before you do, you will get the blame for this crime and your accomplice will go free. If you both hold your silence, we can still prosecute you for some minor crimes. The detectives are hoping that both suspects will turn each other in at the same time and they will both get jail time for their crimes.

Another way to present this dilemma is in the following form:

		Player 2	
		Collude	Cheat
Player 1	Collude	20, 20	0, 30
	Cheat	30, 0	10, 10

If Player 1 assumes that Player 2 is untrustworthy and prone to cheating, then Player 1 can minimize his/her losses by cheating as well. If Player 1 assumes that Player 2 will be faithful to the deal and hold his/her silence (collude), then Player 1 can't help noticing that his/her payoff would be better if he/she cheats (i.e. blames the accomplice for the crime) and Player 2 does not cheat. Player 2 has the same realization. The end result is that both players are tempted to cheat and usually both go to jail.

Under what circumstances would the accomplices resist the temptation to turn on each other?

Researchers have found that in a one-shot version of this game the equilibrium is that both players cheat. In an iterated game, though, it is possible for the players to collude with each other. By playing this game over and over again, the two players can train each other to cooperate.

In the depictions of this situation on television, the suspect that resists the temptation often mentions that their accomplice has a network of fellow criminals who could punish them even if the accomplice goes to jail.

The Prisoner's Dilemma analogy has been applied to many situations, but one of the most famous applications is to the study of global nuclear strategy.

		USSR	
		Cooperate	Attack
USA	Cooperate	20, 20	0, 30
	Attack	30, 0	10, 10

Both the USA and the USSR have large arsenals of nuclear weapons. In a standoff, both are tempted to attack their opponent using their nuclear weapons. However, each country knows the other country also has access to nuclear weapons. If the USSR can get away with nuking the USA and be assured that the USA would not be able to counter-strike, then there is the potential for the USSR to win the Cold War using nuclear weapons. If neither country can give a strong enough signal that they would counter-strike, then both countries are tempted to nuke each other. If the countries can convince each other that they would definitely fire off a counter-strike, then the equilibrium outcome is for both countries to refrain from using their nuclear weapons on each other.

For a more in-depth explanation of PD, see the Stanford Encyclopedia of Philosophy: Prisoner's Dilemma - Standford Encyclopedia of Philosophy

2 Problem Statement

Your task is to code an iterated Prisoner's Dilemma simulator where players will take the following strategies:

1. Always cheat
2. Always cooperate
3. Alternate between cheating and cooperating
4. Tit-for-tat: Choose to cooperate in the first round. If the other player cheats, punish them by cheating in the next round. If the other player cooperates, reward them by cooperating in the next round.
5. Reverse tit-for-tat: Punish cooperation with cheating. Reward cheating with cooperation.

After creating your simulator and testing all combinations of strategies, answer the questions in the "Questions" section.

3 Tools

You will be using the Python programming language and Python interpreter for this lab. The code will be written using a text editor, and the interpreter will be called using the command line.

4 Setup and Programming

4.1 Python

1. To begin, create a new text file and label it prisoner_dilemma.py.

2. Open the text file in the text editor of your choice (such as Notepad or Vim.)

3. Enter the following code at the top of your file:

```
from __future__ import division
```

   (This code will allow Python to handle division in a more intuitive way, doing floating point division by default.)

4. Next, we need to define some variables that will be used throughout the program. Type the following code into your file:

```
COLLUDE,CHEAT = 0,1

''' Payoff Matrix '''
PAYOFFS = (
            ( (20,20), (0,30) ),
            ( (30,0), (10,10) )
          )

''' Strategies Dictionary '''
STRATEGIES = {
            "ALWAYS-CHEAT"   : ((CHEAT,CHEAT),(CHEAT,CHEAT)),
            "ALWAYS-COLLUDE" : ((COLLUDE,COLLUDE),(COLLUDE,COLLUDE)),
            "ALTERNATE"      : ((CHEAT,CHEAT),(COLLUDE,COLLUDE)),
            "TIT-FOR-TAT"    : ((COLLUDE,CHEAT),(COLLUDE,CHEAT)),
            "REVERSE-TFT"    : ((CHEAT,COLLUDE),(CHEAT,COLLUDE)),
            }
```

   The first two items are two aliases we will assign to the values 0 and 1, so that way we have readable labels later on for COLLUDE and CHEAT.

   Next we define a payoff matrix, as a three-dimensional tuple. Using this matrix, we can find what payoff player one will get if he cheats and player two colludes by the following:

```
PAYOFFS[CHEAT][COLLUDE][P1]
```

   where P1 is equal to the value 0. This will return a value of 30. The first offset is player 1's move, the second is player 2's move and the last indicates whether we want the payoff for player 1 or player 2 under those conditions.

   The STRATEGIES dictionary has labels for the various strategies, which map to tuples containing what the next move should be, given the previous choices of player 1 and player 2. The first tuple in a strategy defines the outcomes for when a player last colluded and his opponent colluded or cheated on the previous turn. The next tuple defines the outcomes for when a player cheated on his previous turn and his enemy colluded or cheated, again in that order. For example, consider the TIT-FOR-TAT strategy. Regardless of player 1's choice to COLLUDE or CHEAT on the previous move, his next move is the same based solely on player 2's previous move. Therefore, both tuples are the same for this strategy, namely (COLLUDE, CHEAT). Within the tuple we chose index 0 or 1 (COLLUDE or CHEAT) based on player 2's previous move: when player 2 cheated on the previous move (note that CHEAT = 1), we select the index 1, which gives us a value of "CHEAT" for our next move (as we expect). If player 2 colluded on the previous move (COLLUDE = 0), we select the index 0, which gives us a value of "COLLUDE." As another example, the REVERSE-TFT (Reverse Tit-for-Tat) strategy says that a player cheated and the opponent colluded, the player should cheat on the next turn. In this way, the STRATEGIES dictionary can encode the various choices we should make based on the previous choices of the two players.

5. Next, we will create some functions. Enter the following code, beneath what you've already entered:

```
''' Functions '''
def converged(l):
    return len(l) >= 5 and l[-5]==l[-4]==l[-3]==l[-2]==l[-1]
```

This first function checks to see if our results have converged to a stable outcome. It takes in a list of previous outcomes and checks if two conditions hold:

(a) There are at least five items in the list, and

(b) All five most recent items are equal.

If these two conditions hold, the function returns True; if not, False is returned.

Next, add the following two functions:

```
def simulate(strategyP1, strategyP2, prevP1, prevP2):
    global PAYOFFS, STRATEGIES, COLLUDE, CHEAT
    history = []
    while len(history) < 1000 and not converged(history):
        nextP1 = STRATEGIES[strategyP1][prevP1][prevP2]
        nextP2 = STRATEGIES[strategyP2][prevP2][prevP1]
        history.append(PAYOFFS[nextP1][nextP2])
        prevP1,prevP2 = nextP1,nextP2
    return history

def getOutcome(strategyP1, strategyP2, p1First, p2First):
    results = simulate(strategyP1, strategyP2, p1First, p2First)
    averageP1 = sum([item[0] for item in results]) / len(results)
    averageP2 = sum([item[1] for item in results]) / len(results)
    return (converged(results), results[-5:], (averageP1, averageP2))
```

Notice, you must indent your code as shown, or you will get an error. (Python is whitespace sensitive.)

The function, simulate(), is what performs our actual simulation of the iterated prisoner's dilemma. It takes four parameters:

- strategyP1 - This is the name, as a text string, of the strategy player one will use. For example, you would use "ALWAYS-CHEAT" for the Always Cheat strategy.
- strategyP2 - This is the name, as a text string, of the strategy player two will use.
- prevP1 - This is the starting condition for player one. Legal values are COLLUDE and CHEAT.
- prevP2 - This is the starting condition for player two. Legal values are COLLUDE and CHEAT.

The function begins by declaring access to global variables we defined outside our functions: PAYOFFS, STRATEGIES, CHEAT, and COLLUDE. It then creates a new list to store our payoff history over our iterations. We then enter the main loop, which continues either until we've completed 1000 iterations or the outcomes have converged.

Within the loop, we calculate the next move for players one and two, using their previous moves and their current strategies. After this, we plug the calculated choices into the PAYOFFS matrix and add the result to our history.

The outcome() function is simply a nice function to do a simulation and calculate some summary statistics, such as average payoffs, final states, and whether or not the outcomes converged.

6. Lastly, we enter the driving code for testing all possible combinations of strategies:

```
''' Test All Strategy Combinations '''
for i, strategyP1 in enumerate(STRATEGIES.keys()):
    for strategyP2 in STRATEGIES.keys()[i:]:
        print "\n---------------\n"
        print "Player 1 Strategy:", strategyP1
        print "Player 2 Strategy:", strategyP2
        outcome = getOutcome(strategyP1, strategyP2, COLLUDE, COLLUDE)
        print "Converged?", outcome[0]
        print "Final State", outcome[1][-1]
        print "Avg. Payoff P1", outcome[2][0]
        print "Avg. Payoff P2", outcome[2][1]
```

   This code consists of two nested loops, where the outer loop iterates over the possible strategies for player one, while the inner loop does the same for player two. We perform a simulation for each strategy combination and print out the results to the screen.

7. Save the file and open the command line (From the Windows menu bar: Start > Run... > cmd) Navigate to the folder where you created your file (use the following command: cd "C:\Folder\where file\is".)

8. From the command line, type the following to run your program and hit enter:

```
python prisoner_dilemma.py
```

5 Questions

1. Which combination(s) of strategies produce(s) an equilibrium of cooperation? Why?
2. Which combination(s) of strategies produce(s) the highest payoffs? Why?
3. Which combination(s) of strategies lead(s) to mutual cheating?
4. Do all strategies converge (i.e. settle down to the same outcome for five iterations)?
5. Test different combinations of start conditions ((collude, collude), (collude, cheat), (cheat, collude) and (cheat, cheat).) Are there any differences in the eventual outcome convergences?
6. For each combination of strategies, report the equilibrium outcome (i.e. when the outcome is the same 5 times in a row) by filling out the following table:

Player 1/Player 2	Always Cheat	Always cooperate	Alternate	Tit-for-tat	Reverse tit-for-tat
Always Cheat	*cheat / cheat*				
Always Cooperate		*cooperate / cooperate*			
Alternate					
Tit-for-tat					
Reverse tit-for-tat					