Cooperative and Competitive Learning Agents in games with Non Pareto Optimal Nash Equilibria

December 18, 2018

1 Introduction

Game theory is the study of situational analysis and cooperation and competition. A game is defined to be a set of players and strategies and payoffs. Depending on what strategies are chosen, the payoffs to each player are different. Although not the classic sense of game, many games and real life scenarios (like the Cold War with Russia) can easily be expressed in this format.
Although many of these games are easily solvable, many spark interesting psychological and philosophical arguments about human nature such as the Prisoners Dilemma (or generalized to the tragedy of the commons) and are heavily researched in the fields of mathematics, statistics, economics, and other social sciences. Thus, by no means do we have a complete understanding of how to interact with these seemingly simple games and how their study can be used in everyday life.

More pertinent to the field of Artificial Intelligence is the discussion surrounding independent self-learning agents and their ability to cooperate with others to maximize society's well-being, rather than being selfish to maximize their own well-being. These types of situations were discussed in class during the Ethics lecture, where the thought problem of designing a fleet of autonomous emergency vehicles was posed. How would these vehicles interact with each other to maximize society's well-being?

In this paper we pose two separate "games" to play, one which naturally encourages competition and one which encourages cooperation, and observe the interactions between players in these situations.

The competitive game, which we call the Extended Prisoner's Dilemma (EPD), resembles the Prisoner's Dilemma but is extended to three possible strategies (cooperate, neutral, and defect), and any number of players. In this game, defecting against cooperating players leads to higher personal utility but lower societal utility. To play this game, we defined several set-strategy players, such as a Random Player, a Cooperative Player, a Tit For Tat Player, and a Greedy Defector Player. We also implemented an Expert Learner, which determines its strategy through a weighted sum of the other players strategy, increasing weights on successful players over time while decreasing the weight of unsuccessful players [1]. Finally, we developed a variant of reinforcement learning, random reinforcement learning (RRL), to learn a strategy to play this game through the

observation of other players playing the game. The methodology of the RRL agent is described more in detail in Section 4.2.

The cooperative game, based off a game played in Harvard's Psych 15 class and a version of the Tragedy of the Commons, models a situation where cooperating with other players leads to maximum utility for all [5]. This game is explained more in detail in Section 4.3.1. To determine the optimal combination of players to successfully play this cooperative game, we employed a modified genetic algorithm (GA) to adjust cooperation/neutral/defect tendencies of a group of players playing the game together [4].

2 Background and Related Work

There are many groups exploring the cooperative/competitive tendencies of learning bots. The primary research group in the field is Google's DeepMind, which has published research exploring multiple game environments encouraging competition or cooperation.
One exploration is into RL for two games biased towards cooperation and competition. The first is the Gathering Game, where players are trying to collect apples from a pile. They can temporarily disable their opponent with a laser, allowing themselves to gather more apples. The second is the Wolfpack game, in which agents are in a maze with prey and points are not only awarded to the player who captures the prey, but also the agents near the prey. By changing the environmental variables, such as the number of apples available or the computational power of the players, DeepMind has explored the effect of stressing variables on cooperation tendencies of an inherently competitive game [2]. This DeepMind paper utilized Deep Multiagent Reinforcement Learning.

A second DeepMind paper discusses games in a team situation, which is the same category of game situations as the cooperative game we will be exploring in this paper [3]. The result of the study was that the computer agents did better on average than human players and were rated to be more collaborative. This is an interesting result because despite knowing the optimal strategy, at least one section of Psych 15 did not follow this strategy when playing the cooperative game.

3 Problem Specification

Our problem specifies two tournament-like scenarios playing a game we created which we dub the Expanded Prisoner's Dilemma which has many of its properties, but with 3 strategies (defect 0, neutral 1, cooperate 2) and can have an arbitrary number of players. This game stores a vector of players $P = [P_1, \ldots, P_n]$ and on every turn, each player submits a strategy $s_i \in \{0, 1, 2\}$ and based on how each player acts, they receive a reward. The reward is based on primarily the societal value. The more players that cooperate, the better the society does and the more points are given to the players on average. However, within the players, those who defect steal points from those who cooperate. The societal score, k, is the sum of the strategies so it is $k \in \{0, 1, \ldots, 2n-1, 2n\}$ and the total number of points is evenly distributed from -100 to +100 with a societal score of 0 meaning -100 and $2n$ meaning +100. Mathematically, the number of points to be given is explicitly $\frac{100k}{n} - 100$. Neutral players get the average. A bonus is generated that is stolen from cooperators to give to defectors (if there are no cooperators or defectors, the bonus is taken/given to neutral

players instead) that is equal to $25 + \left\lfloor \frac{100}{1+k} \right\rfloor$. Thus, cooperating is dominated by defecting.[1]

Using this game, we sought to develop an artificially intelligent agent to learn how to play against a set of opponents through reinforcement learning without revealing its strategy. This would be especially valuable if its opponents are trying to develop a strategy against it at the same time. This allows it to not reveal its strategy and learn to play against them. The second goal was to see if we could get a set of robots to *cooperate* to feed a certain player all the points if they were to be judged by their best player.

4 Approach

4.1 Random Reinforcement Learning

4.1.1 Motivation

Consider the common tournament scenario in which a collection of teams will scrimmage many times before the final tournament which determines their rankings. In said scrimmages, teams will want to not only improve how they play a game, but learn how to play against their opponents. At the same time, their opponents will be doing the same and trying to learn how they play and adapt their strategy against their opponents. This tournament model is accurate to many real life scenarios such as preseason in almost every sport or scrimmages in e-sports and can be expanded to realize many artificial intelligence settings such as automatic vehicles driving next to real life drivers in normal situations being thought of as the scrimmages and when an accident or unusual scenario occurs, this is the tournament that matters. Under circumstances like these, it is beneficial to learn and develop a secret strategy. That is, if I can learn to successfully play against my opponent and deny them the same privilege, I will have an advantage in the final tournament. For terminology sake, let's establish the pre-matches (which have little to no effect on final outcome) as scrimmages and the tournament as the competition that matters and which rates the agents. Normal scenarios would have more scrimmages to matches in the tournament and this will be how we model our agents. In our model, the game will be our expanded prisoner's dilemma played with 6 players. This is the same in both the scrimmages and the tournament.

4.1.2 Learning Assumptions

To solve this problem, we implemented a variant of reinforcement learning we call random reinforcement learning. A standard reinforcement agent uses Q-learning meaning it stores what is

[1] Due to the fact that there are 3 strategies and points are stolen/given from neutral players if there are no cooperators or defectors but are the other, neutral and defect are not pure dominant over each other, but defect does completely dominate over cooperate. This is where it differs from prisoner's dilemma in that the Nash Equilibrium is a mixed strategy between neutral and defect depending on the number of players, but it is still **not** Pareto optimal since everyone can always gain if every defector switches to neutral for example and if all neutral, they switch to cooperate. This can easily be seen by going through a couple of examples. To do this, run the first 2 lines of code in the tester routine with different numbers of players to generate the game tensors. See section A in the appendix for more details on how to run this.

learns to be the Q-values of its scenario and as it learns, it explores to figure them out and then exploits what it knows as good strategies. According to our model, there is little reason to exploit during the scrimmages as they are not how the agents are graded. This is where our agent will learn how to properly play against its opponents. In this game, there is no purely dominant strategy, however, the *cooperate* strategy is dominated by defect. According to the rational opponents paradigm, this would mean that a rational agent should never play this strategy which is true in a single game setting, but since the Nash Equilibrium of *defect* is not Pareto Optimal (meaning there exists other states in which **all** players improve their utility) building trust over multiple games can lead to improvement for all (we call this societal gain when the average utility rises). Thus, a proper learning agent will learn how its actions affect its opponents and when it is possible to build trust and fall into a Pareto Optimal equilibrium or when your opponent won't allow this and playing according to the Nash Equilibrium is the best choice.

In order to have the same probabilistic guarantees on the convergence of the reinforcement learning strategy (completeness and correctness) while using the random approach, some assumptions must be placed on the strategies of the players. For the basic non-learning agents, to develop their strategy, they are only allowed to look finitely back through the game player and in our case, we limited this to the turn before. Thus in order for one of these agents to implement their *strategy*, their only inputs were either hard coded (to give them a personality such as greedy or cooperative) and what every player's strategy was the last turn. Furthermore, we imposed noise on every player meaning that even if they decided on a given strategy, it is possible with some probability $\epsilon > 0$ that any other strategy was chosen as well. This means that during the scrimmage portion (which is when the reinforcement learning agent trains and solves for its Q-scores) if it plays randomly, any Q-state can go to any other. This can easily be seen by considering the Markov Matrix of Q-states. For the non-learning agents, by our assumptions of noise mean that any vector $S = [s_1, \ldots, s_n]$ has a probability to move to any $S' = [s'_1, \ldots, s'_n]$ where s_i is the strategy that player i just chose and s'_i is the strategy they will employ the next turn of the game. When we adjoin our reinforcement learning agent as player 0, since all other players P_i $i > 0$ will maintain this property that any strategy s'_i is achievable in the next turn after strategy s_i. This means that as the number of scrimmages approaches infinity, every Q-state transition will occur infinitely many times and since this progression is Markovian (by our assumptions), we can employ standard geometric discounting and prove that this will converge. This is equivalent to saying that this Q-state Markov matrix is ergodic which we know is true since every Q-state leads to any other which according to Markov's theorem makes this matrix ergodic. Thus, our reinforcement agent will eventually converge to the best strategy against its opponents. If we consider learning bots playing against this, they will not be able to isolate the strategy that this reinforcement learning agent is developing and they will perceive it as only playing randomly and cannot learn against it. Notably, if we pair only these agents against each other, since they are playing randomly, each will perceive only random actions and thus converge to the Nash Equilibrium of the game.

4.1.3 Implementation

First we instantiate a set of 5 player for which our reinforcement learning agent will play against. These players are $Players = [P_1, P_2, P_3, P_4, P_5]$ and can be any assortment of the trivial or expert learner players created. We will have it play against combinations of different opponents to assess how it performs in different circumstances and demonstrate that it is indeed able to learn a differ-

ent strategy against different opponents. First a test match is done with P_0 set to random player as a baseline. After this, training begins and we play 500 matches of 1000 turns each and feed this data as it happens to the learning agent. The learning agent keeps a table of Q-scores first all initialized to 0 and updates them with the standard biased averaging function:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(R(s,a) + \gamma \mathsf{max}_{a'}Q(s',a')$$

Our agents had a learning rate of 0.2 and a discount rate of 0.8. Since this agent doesn't have a concept of escaping the MDP since this game has no end, when the max magnitude Q-value got too high (above 1000000), they were rescaled such that the max had magnitude 100. This allows us to train with a lot of data and not lose precision as the Q-values get too high. After training is completed, the reinforcement learning agent competed in a regular match of 1000 rounds to assess how well it learned against its opponents.

Algorithm 1 Learning Algorithm

```
procedure UPDATEQSCORE(qState , currReward , nextState)
    key ← key(qState)
    currQS ← self.QStates[key]
    if nextState ≠ None then
        key2 ← key(nextState)
        key2A0 ← self.QStates[key2.append(0)]
        key2A1 ← self.QStates[key2.append(1)]
        key2A2 ← self.QStates[key2.append(2)]
        maxAction ← max(key2A0, key2A1, key2A2)
        update ← (1 − α) ∗ currQS + α(currReward + γmaxAction)
    else
        update ← (1 − α) ∗ currQS + α(currReward)
    end if
    if abs(update) > 1000000 then
        rescaleQValues()
    end if
    self.QStates[key] = update
end procedure
```

In the above pseudocode, qState is the current Q-State complete with the action of P_0 and nextState has only the actions that the other bots $[P_1, \ldots P_5]$ played in the following turn, so appending a 0, 1, or 2 is adding all possible actions for the next turn.

4.2 Genetic Algorithm

4.2.1 Motivation

Almost all games that involve cooperation can be construed as competitive games. The Prisoner's Dilemma marks this perfectly as it doesn't have to be a win-lose scenario, but most agents and players make it into that. The tragedy of the commons is another example in which people consider only the personal gain to be achieved in the game when society can be better off by cooperating. We wanted to determine if agents could be placed into a game such as this where their

societal score is based off the best performer in the game, but the reward is fully societal. This is to say that they compete to see who will represent them, but this *champion's* score is shared by all. Thus in a game such as tragedy of the commons (or our expanded prisoners dilemma) the solution is simple: choice a single player and feed them all the points. The problem comes in choosing this alpha player and forcing everyone else as a beta player designed not to help themselves, but maximize this other players utility. This framework is inspired by the Harvard class Psych 15 which played a similar game amongst their sections. Each section played the tragedy of the commons and the winners of each was compared and the highest score received a monetary prize which may be shared with the whole section. Thus, in this case, the competition is not actually internal to the section, but between them so a rational section would **arbitrarily** choose an alpha player as their champion and feed them the score. Even though this guarantees a win, it is in the nature of humans to not do this for it requires blind trust, so we sought to see how we can best create a section of independent agents to converge upon this solution.

4.2.2 Implementation

The genetic algorithm implementation is fairly classical with few variations. A population of sections was created where each section consisted of 6 players. The *genetic makeup* of each of these players was their probability of choosing each strategy. The fitness function was assessed by running 75 rounds of EPD and choosing the winner and adding the lowest possible score + 1 to give every section a positive fitness. Suppose there was a 1% variation, in order for this to propogate and not realize a difference in 75 rounds, it would have to get lucky 75 times (or it would get lucky x times and some opponent $75 - x$ times). This probability of this happening is 47% so the error in assessing fitness this way is only differentiating within 1% between players in different sections. Selection was directly proportional to the calculated fitness function. Cross-over was done in two ways. The primary cross-over was the classical one where the first n players from the first parent are combined with the last $6 - n$ players from the second parent with a random n. This however can make this example take two excellent parents and create a bad child (see 1), so we used this first to get to close to good solutions quickly, but then switched to a cross-over variant that just averaged each player in the section by averaging their probabilities of choosing any strategy. We switch to this crossover variant when the best player in some generation is within 95% of the optimal solution. The mutation function randomly chooses and player and moves 5% probability over from it choosing a certain strategy to a different one. See figure 1 for more details as how all the data is stored in the code and how cross-over works.

5 Experiments

5.1 Random Reinforcement Learning Agent

In order to quantify the effectiveness of the random reinforcement learning agent, a testing program was written to create every combination of the five players (Random, Cooperative, Defect, Tit For Tat, and Expert Learner) in a 6-player match with 1000 turns and run an experiment comparing the performance of a Random Player, Expert Learner, and RL Agent. In these experiments, a ”Cooperative” player is defined to have a cooperation rate of 0.8 and defect rate of 0.1, while a ”Defect” player is defined to have a cooperation rate of 0.1 and a defect rate of 0.8.

For each match, the first player was always set as Random. Then, the program followed the typical training sequence for the Random Reinforcement Agent: playing a game with the random as a baseline, training the RL Agent, testing the RL Agent, and playing an additional game with an Expert Learner. The results were then stored in a CSV file.

5.2 Genetic Algorithm

We decided to use a Genetic Algorithm for this optimization problem of finding the ideal combination of players because the representation fit very perfectly into the scheme of a GA: each member of the population of candidate solutions has a set of properties (the players) that can be altered. This quality of the optimization problem made a GA a better option than, say, simulated annealing.

Our GA had a variation on the classic GA's "crossover", which would normally involve picking an index and swapping the players after that index in that specific game setup. This is due to the problem of high-fitness game setups with the dominant player in different index positions, as shown in Figure 1.

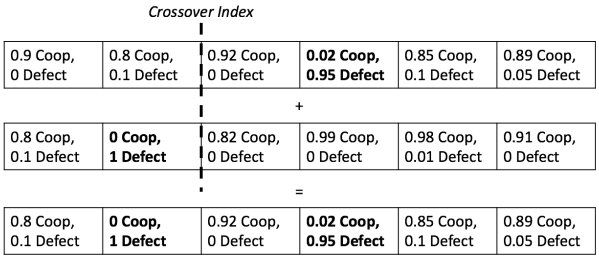

Figure 1: Example of a situation where classic crossover of two high-fitness parents can result in drastically sub-optimal children. The ideal section contains 5 players with 1 Cooperate and 0 Defect and 1 player with 0 Coop and 1 Defect. This creates the highest possible score for the Defecting player. Both parents here are very close to that, but their child is far. Note: the probability for the neutral strategy is implicit by solving for the missing probability after summing the probability of cooperating or defecting. i.e the first player in all three sections has 0.1 probability for choosing neutral.

To solve this problem, the crossover method was updated to include the situation where, if one of the parents were within 5% of the optimal fitness value (where one player defected and all the others cooperated), then the values were averaged in the crossover step.

The implementation of our genetic algorithm required two major hyperparameters, population size and mutation rate, which determine the number of generations till convergence. Therefore, two experiments were run: one with variable population size with a fixed mutation rate of 0.09, and one with a variable mutation rate and fixed population size of 250.

We can expect that increasing the population size leads to convergence in a fewer number of generations, since there are more individuals in the population being crossed and mutated. The relationship between increasing mutation rate and generations till convergence is interesting be-

cause a higher mutation rate would bring the population closer to convergence in a fewer number of generations but may take more generations to fine-tune a high-fitness population to produce one individual with optimal fitness.

6 Results

6.1 Random Reinforcement Learning Agent

The experiment detailed in Section 5 was run twice and averaged. The full data files are named "PlayersOne" and "PlayersSecond". The results showed that the RRL agent performed, on average, 9261 points better than a random agent in its place. The RRL agent performed on average 2677 points better than an Expert Learner in its place, and beat the Expert Learner 70.63% of the time. The RRL agent won its match 61.9% of the time.

A graphical view of a part of this data is presented in Figure 2, where the games with only cooperative and defecting players were compared.

Number of Cooperative Players	Random Player Societal Utility	RRL Agent Societal Utility	Expert Learner Societal Utility
0	-29037	-37314.25	-40578
1	-7468	-23991.5	-18154.5
2	9166.5	-11785	-2733
3	25746.5	6025	15832.5
4	44102	35497.5	32746.5
5	57711.5	64962	63080

Figure 2: Heatmap of the societal utility values from the RRL experiments with only cooperative and defecting players.

6.2 Genetic Algorithm

The two experiments concerning the GA (detailed in Section 5) were run twice, and the results were averaged together. The files containing the output data are named "GAoutput1" and "GAoutput2." Both times, the solution converged to the optimum before 3,000 generations, which was the cutoff for termination of the algorithm.

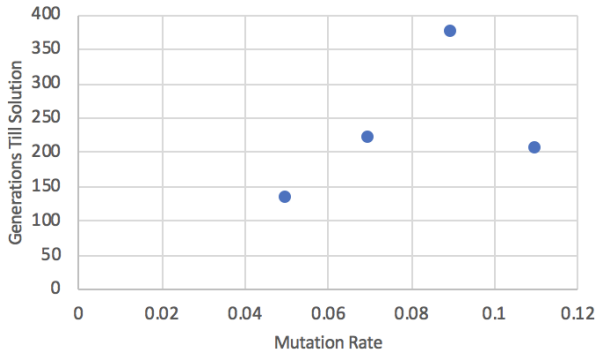

Figure 3: Graph of the results of the GA experiment with a changing mutation rate.

As shown in Figure 3, the relationship between mutation rate and generations is not straightforward. A lower mutation rate seems to be more favorable in terms of convergence.

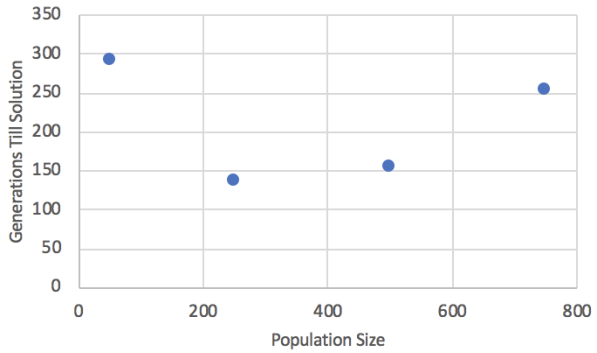

Figure 4: Graph of the results of the GA experiment with a changing section size.

Unlike what we predicted, the relationship between section size and generations is not straight forward either, as shown in Figure 4.

7 Discussion

7.1 RL agent

Implicit data does indeed show that the RL agent was able to successfully learn how to play very successfully against its opponents. Winning most of the time and beating its learning counter part of expert learner shows that in an unknown circumstance, implementing this learning strategy is likely to do better than a nave expert learner. However, figure 4 shows that this agent isn't always the most cooperative. When paired against cooperative opponents, it does find the cooperative solution, however it is pretty grim against opponents that have defectors or other greedy players with it. This means that if one knows that they will be paired with classical rational opponents in a game such as this, RL will perform well against them since they will defect anyway and there is nothing else to be done, however in a game where there is an extra metric of how well society

does, expert learning might fair better. If winning is the goal, RL agent is definitely the best all around agent. In situations where you care about how society does, the RL agent converges to a policy that is too greedy and should not be used.

7.2 Genetic Algorithm

As expected, the genetic algorithm was always able to converge to the optimal solution quickly. As far as minimizing regret grows, the data shows that a lower mutation rate is generally better as it may not be screwing up good results. A high mutation rate of 0.11 did better probably due to the large population meaning that for the high amount of bad mutations it did, it may have done enough good ones as well that the selection process could differentiate them in the next one. It is most likely this is a local minimum and increasing the mutation rate further would make it take longer to converge again.

For population size, the expected trend would be a strong negative correlation between population size and generations till solution. We see this in the increase from 50 to 250, but after that it takes longer. This is most likely due to the second cross-over algorithm. When this averages two good sections, it could bring 2 parents both very close to perfect further away by taking 2 players away from having a 1 and a 0 for their values (as is in the optimal solution) to something close. After this, it would need to randomly mutate to fix it. If the population is large enough, two equally good parents can again create less good children and with a larger population, the probability of this case occurring is higher. Thus, this is the problem with the second cross-over algorithm as opposed to the classical first one. To fix this, we could employ a "sorting mutation" instead of switching the cross-over algorithm. This mutation could swap the positions of players and this combined with the first cross-over algorithm would eventually swap all the defecting players to the same position and get rid of the original problem. Implementing this would likely make the correlation continue that as population increases, the species evolves faster.

A System Description

The file necessary to run the project can be found in Tester.py. The main method at the bottom of the file contains 4 lines of code:

- The first two demonstrate the generation of a game grid with two players and prints it out. The number can be modified to demonstrate that our program can automatically generate payoff matricies for any number of players.
- The third line, GA(), runs the genetic algorithm experiment detailed in Section 5.2.
- The last line, matchCombinations(), runs the RRL experiment testing every combination of players with the RRL Agent learning scheme detailed in Section 5.1.

These three commands encompass the major functions created for this project. The classes created for this project include:

- Player: Contains the code for the implementation of the 3 fixed-strategy players (RandomPlayer, SetPlayer, and TitForTatPlayer), the Expert Learner, and the RRL Agent. Each of these players are classes which extend the abstract class Player.

- Game: Contains the code for playing one round of a Match and calculating the payoff. The Game can be played using an Extended Prisoner's Dilemma, or another pre-defined payoff matrix.
- Match: A Match holds Players and a Game and runs this game for a certain number of turns.
- Genetic Algorithm: Holds the code for running the GA.

References

[1] Mit lecture 09.520: Online learning. 2008.

[2] Thore Graepel, Janusz Marecki, Joel Leibo, Vinicius Zambaldi, and Marc Lanctot. Understanding Agent Cooperation.

[3] Max Jaderberg, Wojtek Czarnecki, Iain Dunning, Luke Marris, and Thore Graepel. Capture the Flag: the emergence of complex cooperative agents | DeepMind.

[4] S. Mangano. *An Introduction to Genetic Algorithm Implementation, Theory, Application, History and Future Potential*. 1991.

[5] Robert L. Trivers. The Evolution of Reciprocal Altruism. *The Quarterly Review of Biology*, 46(1), March 1971.