Multiple Players Iterated Prisoner's Dilemma Implementation

Daniele Foscarin
Information Engineering Department
University of Padua
Padua, Italy
daniele.foscarin@studenti.unipd.it

Giacomo Loreggia
Information Engineering Department
University of Padua
Padua, Italy
giacomo.loreggia@studenti.unipd.it

Donald Shenaj
Information Engineering Department
University of Padua
Padua, Italy
donald.shenaj@studenti.unipd.it

Nicola Cibin
Information Engineering Department
University of Padua
Padua, Italy
nicola.cibin.1@studenti.unipd.it

***Abstract*—The Prisoner's Dilemma (PD) is a classical game analyzed in game theory, which is widely used to model social/economical interaction. In the classical formulation of the game, we have two players facing the decision to collaborate or defect, and given the choice of each prisoner we can obtain the payoff that they get. There have been a lot of experimental studies on evolving strategies for 2-player Iterated Prisoner Dilemma. However, there are many real world problems which cannot be modelled with just two players. For this reason, in this paper we tried to extend the game to a n-player scenario, that is a more realistic and general formulation which can model those problems. Furthermore we studied how the system reacts when the population is increased by adding new players who follow a given strategy depending on the results that various strategies achieved in the previous iteration. In the last part of the paper we also extended the reasoning in a context where strategies are allowed to mutate; the goal is to simulate the effect of genetic mutations and the effect of natural selection.**

***Index Terms*—Game Theory, Multiplayer Iterated Prisoner's Dilemma, Changing Strategies, Learning Players, Python Implementation.**

I. Introduction

The Prisoner's Dilemma (PD) is very well know Game Theory case of study [1]. In this paper we will use the following basic linear algebra framework in order to obtain the players' rewards. Given the payoff matrix M, that quantifies the revenue that each player gets depending on whether he cooperate or not, the i player reward can be computed as:

$$r_i = u_i^T M u_{-i}$$

where:

$$M = \begin{pmatrix} 2 & 0 \\ 3 & 1 \end{pmatrix}$$

$$u_i = \text{i-th player move}$$
$$u_{-i} = \text{adversary's move}$$

$$u_x = \begin{cases} \begin{pmatrix} 1 & 0 \end{pmatrix}^\intercal & \text{when player } x \text{ collaborates} \\ \begin{pmatrix} 0 & 1 \end{pmatrix}^\intercal & \text{when player } x \text{ defects} \end{cases}$$

In the case of a single round game, the outcome is trivial given that defecting is the strictly dominant strategy and so it guarantees that the outcome will be a Nash Equilibrium; thus each player will choose to defect even if the overall revenue is not the highest obtainable. If instead we extend the game to an iterated interaction scenario, i.e. the static game is played various times, more interesting results could be obtained because some collaborations dynamics could arise. The remainder of this paper is organized as follows. In section III we study how different strategies behave when applied in a Iterated Prisoner's Dilemma (IPD) context. In section IV we implement a multiple players IPD (MPIPD) where several strategies play against each other in a round-robin scheme. In section V we iterate what done in section IV by increasing the population implementing a given strategy depending on the results that the strategies achieved in the previous iteration. In section VI we extent the discussion to a rMPIDP were strategies are allowed to mutate. In section VII the work is concluded.

II. Implementation strategies

The definition of the problem leads naturally towards an object oriented implementation. We defined the abstract class

Player that declare the abstract method move() and the necessary methods for the update of the reward and resetting the player. A set of classes that extends the Player class implement every strategy that we considered in this project by defining the move() method with a certain strategy and by the past moves, rewards obtained and any other information that is useful for the decision of the move or for the evaluation of its performances. In this way every player instance is a self contained object that can play in every scenario of the Iterated Prisoner's Dilemma that we implemented.
The implemented players are the following:

- NiceGuy, it always outputs the cooperate move;
- BadGuy, it always outputs the defect move;
- KBadGuy, it outputs the defect move with probability k/100. It is a generalization of the NiceGuy, the BadGuy and every strategy between them;
- Tit4Tat, its first move is cooperate, the subsequent moves are the same as the last opponent move given in input;
- Tit4TatMP, it is a generalization of the Tit4Tat, that works with any number of players by choosing the cooperation move if a certain fraction of the opponents has cooperated;
- GrimTriggerMP, it outputs the cooperate move until a certain fraction of the opponents has defected, from this moment it always outputs the defect move;
- LookBackPlayer, it uses the past rewards to evaluate which move gave him the higher rewards. We can apply a bias on the rewards during the move decision. It chooses randomly if the rewards are equal for the two moves.

After the instantiation of these classes, every game iteration can be easily implemented by calling the move() method for the players involved, computing every reward, and giving to the players their rewards and other required information using their purposely defined methods.

III. Two Player Iterated Prisoner's Dilemma

It is implement a sequence of games, called tournament, between two players. This task is made simple thanks to the self contained player objects, and it only consist in a loop containing the move method for both players, followed by the registration of their reward calculated as described in the section I. We can observe the outcome of various two players tournament in order to draw conclusions about the efficacy of a strategy.
Focusing at first on the Tit4Tat players, we make them play against a KBadGuy and they tends to finish the tournament with a total reward similar to the KBadGuy, but it is impossible for a Tit4Tat to finish with a higher total reward than its KBadGuy opponent. That is explained by the fact that they start by cooperating and after that they will defect the exact number of times as the opponent, since they copy the opponent moves. In a tournament made between two Tit4Tat players, they start by cooperating and they continue to cooperate, reaching the maximum overall reward (defined as the summation of the player rewards).
Focusing on the KBadGuy, we expect that the player with

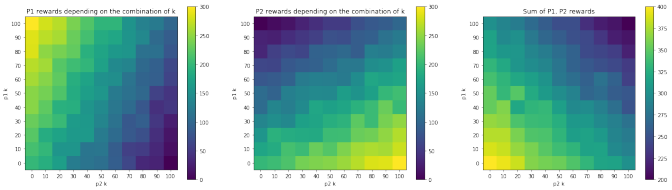

Fig. 1: Final rewards in a series of tournament exploring the entire range of the KBadGuy players

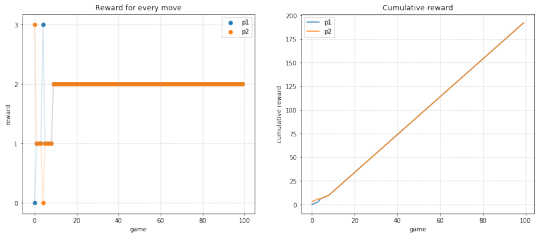

Fig. 2: Rewards in a tournament between two LookBack Players.

higher k tends to win, in order to confirm this we want to perform a series of tournament using every combination of k parameters in the two players. So we make the k parameters range between from 0 (that is a NiceGuy strategy) to 100 (that is a BadGuy). We want to observe the individual rewards and the overall rewards. In Figure 1 we can see that the individual reward is higher when the player has $k = 100$ and its opponent has $k = 0$, but the overall reward is higher when the two players adopt a cooperative behaviour. In this way we illustrate the statement made in section I.

We wanted to implement a player that adopt a more complex strategy, in order to have high probabilities of victory against a wide range of KBadGuy and also to try to develop a cooperative behaviour that maximizes the overall reward. The LookBack Player chooses as the next move the move that made it get higher rewards in the past. If the rewards are equal or it is the first move it chooses randomly. During the decision process we add a bias term to the values of the past rewards in order to simulate a payoff matrix with zero average. It is not immediate to foresee what is the outcome af a tournament between two LookBack Player, but after few moves at the beginning, they learn that the cooperation is the best rewarding strategy and they finish the tournament with a similar total reward by cooperating. An example of this behaviour is displayed in Figure 2

We want to test the LookBackPlayer against the entire range of KBadGuy players. So we implement a series of tournament where we vary the k parameter of the KBadGuy. A new LookBack Player is initialized at every tournament so every tournament is independent from the past ones. We perform this experiment 50 times in order to obtain reliable results reducing the effect of the randomness in the players decisions. In Figure 3 we can see that the LookBackPlayer is able to win against every KBadGuy with a k parameter

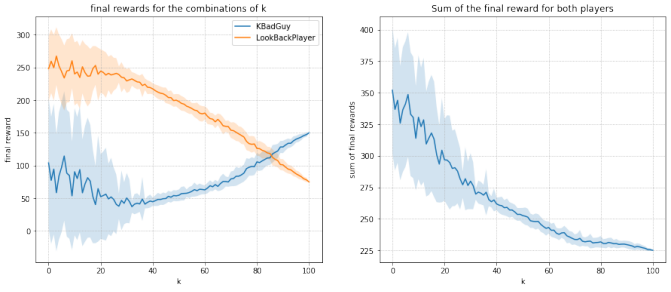

Fig. 3: Averaged result on multiple runs of the LookBack Player vs every KBadGuy. The shaded area represents the standard deviation.

lower than circa 85. We can also see that the LookBack Player produce a rather complex behaviour that depends on its opponents, while playing against a KBadGuy with low k (that is MainlyNiceGuy) its rewards are characterised by a high standard deviation, while with the increasing of k its results are more homogeneous and it is unable to win against a BadGuy ($k = 100$). In this test, as before, the overall reward is higher when the k parameters is lower.

IV. Extension To Multiple Player Scenarios

In order to extend our implementation to a Multiple Player Scenario we decided to follow the framework proposed in [2]. The following is the payoff matrix of the $n-$player Prisoner's Dilemma game, where the following conditions must be satisfied in order to not distort the game meaning:

$$\begin{cases} D_i > C_i \\ D_{i+i} > D_i \\ C_{i+1} > C_i \\ C_i > (D + C_{i-1}) \end{cases}$$

Number of cooperators among the remaining $n-1$ players

$$\text{Player } i \begin{pmatrix} C_0 & C_1 & C_2 & \dots & C_{n-1} \\ D_0 & D_1 & D_2 & \dots & D_{n-1} \end{pmatrix}$$

A large number of values satisfy the requirements of the previous matrix. As in [2], we choose values that, if n_c is the number of cooperators in the $n-$player game, then the payoff for cooperation is $2n_c - 2$ and the payoff for defection is $2n_c + 1$. Furthermore, with this choice, simple algebra reveals that if N_c cooperative moves are made out of N moves of an $n-$player game, then the average per-round payoff a is given by:

$$a = 1 + \frac{N_c}{N}(2n - 3)$$

Number of cooperators among the remaining $n-1$ players

$$\text{Player } i \begin{pmatrix} 0 & 2 & 4 & \dots & 2(n-1) \\ 1 & 3 & 5 & \dots & 2(n-1)+1 \end{pmatrix}$$

In this way, we don't need to actually create the $2 \times (N-1)$ matrix but we can directly calculate the player payoff with a very efficient procedure. The proposed Python implementation is very flexible. In fact, it accepts in input any number of players, any player type and any number of game iterations. In output we will obtain the players playoffs in each round and the moves history.

V. Repeated MPIPD

In the Repeated MPIPD (rMPIPD), multiple games are played one after the other, but every time one player is added. The purpose is to study how the game changes as different types of players join. In this context we call every single game a *tournament*. We could imagine the first tournament of the rMPIPD as a group of people composed by different types of players seen in section II playing the MPIPD game. As the game runs there is a person watching the progress who is supposed at the end of the tournament to remember all the partial moves of the players and to know the final payoffs of the players. After the game ends this person joins in, playing as one of the players that are already in the group. After this first tournament, the procedure is repeated for a certain number of times and the successive persons joining the game do their evaluations based on the means of the trends of the players already in the game that implement the same strategy.

Different criteria can be used by the new players to decide how to play. Anyway, the criterium is decided at the beginning of the rMPIPD and then it is kept the same at every tournament. We can subdivide every criterium into a *macro-criterium* and a *micro-criterium*. In other words, firstly(macro-criterium) a new person joining the game chooses among the other types of players its type based on one of these:

- Maximum final reward
- Random
- Minimum final reward

Then, since implementing a non-random macro-criterium can lead to a tie between the player trend means, a micro-criterium among the following is used:

- Highest number of cooperations
- Random
- Highest number of defections

As one can expect, this type of game is heavily influenced by the initial conditions, i.e., the initial list of players, since every new player chooses its way of playing among the ones already in the group and not on players that are not present.

Different scenarios are built and the results retrieved show the players added time by time and the mean final reward trend per type of player. We present here only some of these results, the ones we investigated and that we think are the most meaningful.

In figure 4 we see the graphs built on an rMPIPD played by a complete set of players. The initial number is the value of the parameter associated to that type of player and chosen for it. The criterium used to add new players is min_r, i.e.,

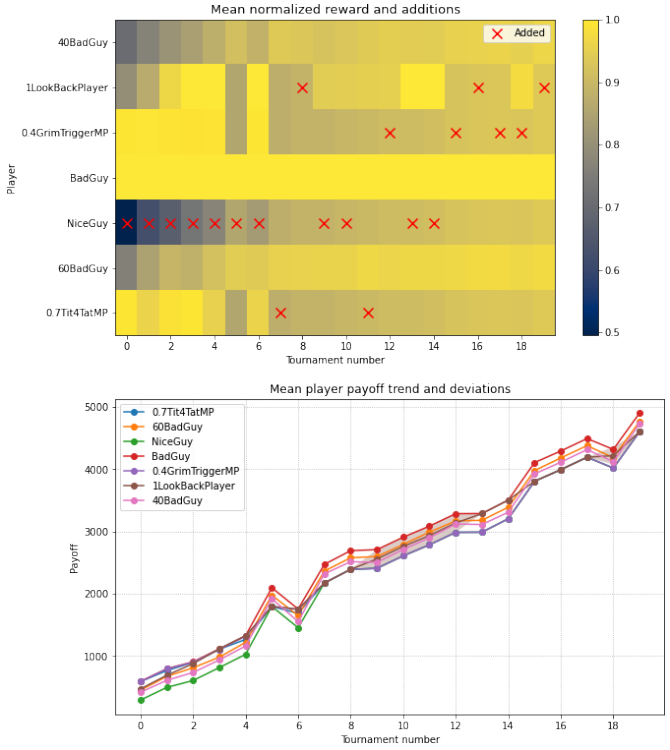

Fig. 4: Analysis on an rMPIPD played by a complete set of players. The criterium used to add new players is min_r, i.e., minimum(macro) and then random(micro).

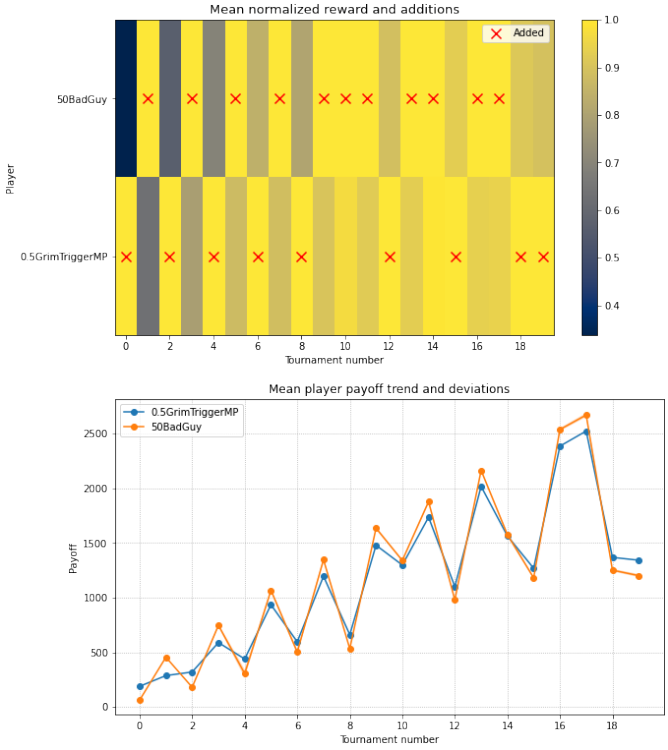

Fig. 5: Analysis on an rMPIPD played by a set of 2 players. The criterium used to add new players is max_c, i.e., maximum(macro) and then cooperation(micro).

minimum(macro) and then random(micro). Surprisingly, the NiceGuy is not the one that always gains the lowest payoff. It's true that there's an initial phase in which the NiceGuy actually gains the lowest payoff. Note that in this phase the 0.4GrimTriggerMP player together with the BadGuy and the 0.7Tit4TatMP achieve the best reward. However, once we add some NiceGuy players these players, who gained the highest payoff, change their behaviour. In particular, the GrimTrigger player doesn't behave in a bad way anymore, since it is not triggered: the number of defecting players is no more enough to trigger him. The Tit4Tat player starts cooperating as well. Generally, the BadGuy and KBadGuys are never added, as they never gain the lowest payoff. Overall, we see that the payoff keeps increasing for all the players. This can be understood also recalling the previous section, in which we have seen that as the number of cooperating players increase, the other players, be them nice or bad, benefit from these good ones and their payoff increases.

A curious game is one depicted in figure 5, in which we see that the tournament maximum reward bounces back and forth to the 50BadGuy and the 0.5GrimTrigger. In particular, we see that in this context, a guy playing randomly, without any bias, either wins a lot, or it looses a lot. The general reward slowly increases because either the number of players increases, and we have seen that this causes the possibility to gain more, and also the attitude is that of cooperation overall. We can explain the trend obtained in the following way: the only way for the GrimTriggerMP players to win the tournament is that of being triggered by the 50BadGuys, which in turn can happen only if the latter are more in number than the former; otherwise the behaviour of the 50BadGuys have no influence on GrimTrigger, so they cannot trigger them. In this last case they are likely to win because they defect more then the GrimTrigger players, gaining more payoff. Moreover, we note that when the GrimTrigger players are triggered, the general reward drops down dramatically for both the characters, as they no more provide the payoff that was advantageous for both. It is as if the 50BadGuys can be punished by the GrimTrigger players only when they are too much in number.

VI. rMPIPD with evolving strategies

At this point we want to simulate the effect of genetic mutations and natural selection in the contest of a repeated MPIPD. Thinking about a parameter that encodes the attitude of an individual to cooperate, we can consider a set of KBadGuy players where the k parameters can mutate and be transmitted to other player, until the best fitted strategy is determined.

For this task we explore the idea of players that can generate "children" similar to them and "die" if their rewards are not good enough. So we implement a function that calls the MPIPD function described in section IV with a list of KBadGuys players, then the worst performing players gets eliminated until their number is not higher than the maxi-

mum population. The remaining players will generate a fixed number of children that inherited their parent k parameter varied by a random delta. Doing so at every repetition we get a new generation of players that play among the old ones, and the worst players elimination prevent them from generating children similar to them. We expect to see the predominance of the players with higher k over the repetitions. We performed this test initializinng the function with 5 KBadPlayer with random k, 50 epochs, max population equal to 40 and 2 children generated at each epoch. The distribution and frequency histogram of k of the last epoch is visibile in Figure 6. As expected we can notice that the value of k, initialized randomly, increases up to 100.

In order to generalize better the effect of natural selection we adapted the function in order to use various type of players and imposing the children to have the same parameters of the parent which produced it.
We test a scenario starting with four players: KBadGuy with $k = 45$, Tit4TatMP with threshold $= 0.5$, GrimTriggerMP with threshold $= 0.5$ and a LookBackPlayer, running for 20 epochs and using the same number of children and the population control as before. In Figure 7 is displayed the last frame of the animated bar chart containing the number of alive player of each type at the last epoch, and we can notice now that in this test only instances of the LooKBackPlayer survive at the end of the game. While looking at the move history of those surviving players, we see that this formulation of the IPD problem has rewarded the most defective behaviour, in fact the remaining LookBack players have adopted a strategy identical to the BadGuy behaviour, by defecting at every move. At this point we can affirm that if we simulate the "evolutionary pressure" as we did in this section, the defecting behaviour strongly emerges as the most convenient for the survival. Anyway, even if this result fits the theory about the Prisoner Dilemma, we do not see the development of cooperating behaviours, it would be interesting to understand if this is because in the current formulation of the evolving-MPIPD problem we do not consider the fact that strong cooperating behaviour could change the effect of the population control, allowing more individuals to stay alive, mimicking what happens in real communities.

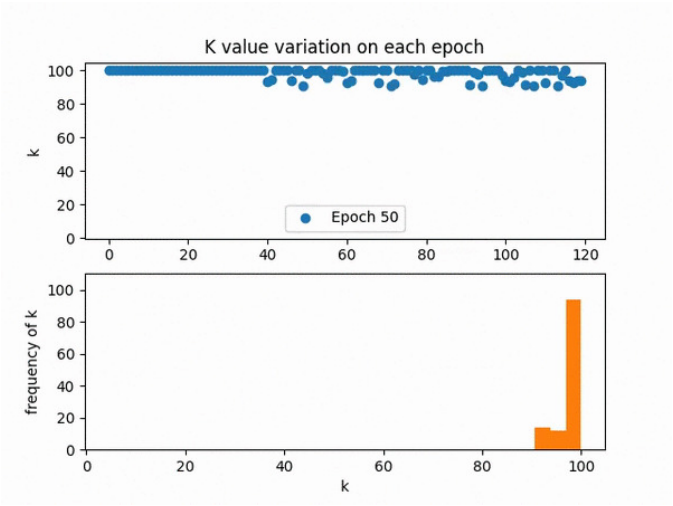

Fig. 6: Distribution and frequency histogram of k at the last epoch.

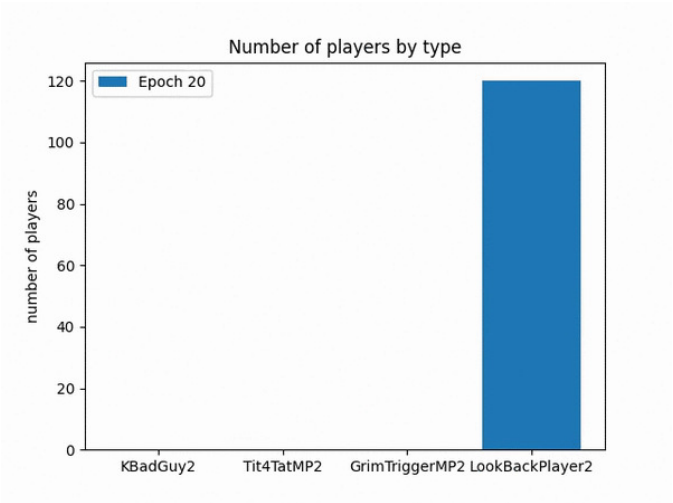

Fig. 7: Last frame of the animated bar chart displaying the number of alive players of each type at the last epoch.

VII. Conclusions

We have built a framework for the implementation of the Iterative Prisoner Dilemma, testing various scenarios about two players tournament, tournaments with an arbitrary number of players, repeated tournaments with an increasing number of players and scenarios about the simulation of an evolving population of players. During this exploratory task we have found out that the LookBackPlayer strategy is well performing in a wide range of situations. In the repeated-MPIDP scenario we saw that the result of interactions between different player strategies is not easy to foresee, but by increasing the number of cooperative leaning players makes the overall rewards increase as well. The defection behaviour is affirmed in the rMPIPD, where we see that the people that mostly defect can

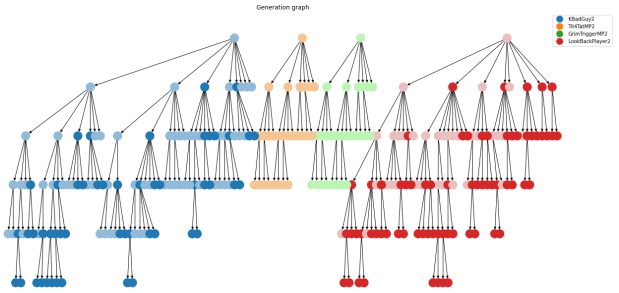

Fig. 8: Tree graph displaying the relationships between the player at the end of the fifth epoch

gain more, while the players that behave well usually gain the lowest payoff, even though the latter can provide the payoff for the others and in some way give them more and more sources as they increase in number.
Finally in the evolving-MPIPD scenario we assisted to the affirmation of the defection behaviour as the answer to the strong evolutionary pressure implemented as the elimination of the less successful players.

REFERENCES

[1] PRISONER'S DILEMMA. URL: https://en.wikipedia.org/wiki/Prisoner's_dilemma, (last consultation: 22/01/2021).

[2] AN EXPERIMENTAL STUDY OF N-PERSON ITERATED PRISONER'S DILEMMA GAMES. *Progress in Evolutionary Computation. EvoWorkshops 1993, EvoWorkshops 1994.*. URL: https://doi.org/10.1007/3-540-60154-6_50, (last consultation: 22/01/2021).