Iterative Prisoner's Dilemma Solution using Genetic algorithm

Nisarg Dave, Data Scientist, Michigan Technological University

***Abstract*—This paper demonstrates the IPD implementation using Genetic algorithm. How we can solve the IPD using the evolutionary computation approach. I did discuss the details on IPD and then how we are gonna apply genetic algorithm for its solution. The next part is to expain detailed process for integrating GA for desired strategic solution. Every minute detail regarding encoding, selection and optimization is discussed in the paper.**

***Index Terms*—Iterated prisoner's dilemma,Game theory, IPD-memory, Genetic algorithm, Evolutionary computation, Genetic programming, Evolutionary optimization, Nash equilibrium**

I. Introduction

THIS paper describes the overall process of applying genetic algorithm for the IPD. Iterated prisoner's dilemma is very famous Game theory problem. The Game theory is center of attraction for many researchers. There are different types of game theory problems such as simultaneous-sequential, cooperative-noncooperative, zero/non-zero etc. The rationality is very important when it comes to game theory. IPD shows how two completely rationals might not cooperate.

A. Prisoner's dilemma

We have two prisoners for the interrogation. Because of the lack of the evidence, the prosecutor gives chance to both the prisoners. They hope to get sentenced for a year, due to the bargain option provided by prosecutor. Simultaneously, the prosecutors offer each prisoner a bargain. Each prisoner is given the opportunity either to: betray the other by testifying that the other committed the crime, or to cooperate with the other by remaining silent.

- each of them betray the other and then they serves 2 years in a prison.
- if A betray other and other remain silent then A will be set free and B will get 3 years in prison.
- if both of them remain silent then they will serve 1 year in prison.

The game theory is all about study of the interactive interaction between individuals in a conflict situation. The Winning matrix is presented below. This is simultaneous, non-zero sum game. I did use Nash equilibrium with the optimum of pareto.

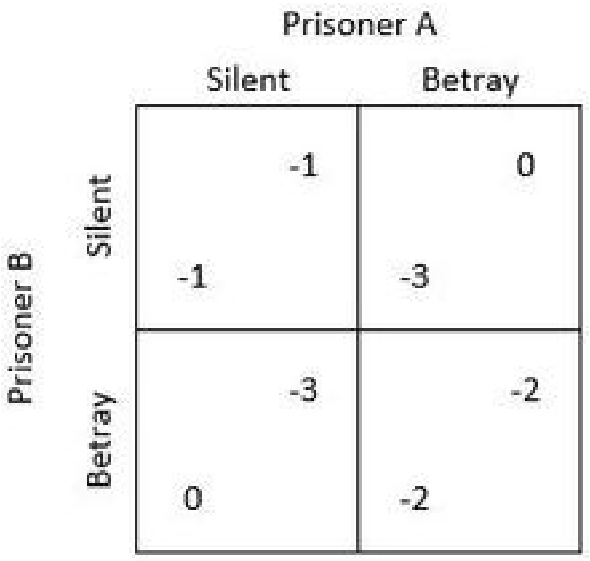
Prisoner A
Silent
Betray
Prisoner B
Silent
Betray
-1
0
-1
-3
-3
-2
0
-2

Fig. 1: IPD Payoff matrix

1) Best strategy: According to me, the best strategy will be "tit for tat with forgiveness" where in the first iteration you just cooperate with the opponent. In the next move player does what the opponent did in previous moves. But sometimes cooperate in anyway with very less probability. According to me this is best one because it incorporates the 4 important characteristics stated by axelrod. They are nice, Retaliating, forgiving, and Non-envious.

2) Encoding for no-memory IPD: When we don't have any history or past behaviour of prisoner then we need 1 bit to encode because we don't have any record or memory so we just need to encode the current action of the prisoner. Prisoner will make the first ever decision and that's either the cooperate or defect so for encoding any one decision we just need 1 bit representation.

3) Basic GA Fitness funtion: For solving the prisoners dilemma, I'll represent the C as 0 and D as 1. let say we have CD in the the representation is 01. For the next input/decision I'll use fitness function as,

$$updated_score = move_1 << 1|move_2$$

so first I'll apply the bitwise left operator and then I'll do the bitwise oring with the new move. Bitwise oring is all about putting 1 where we have compliment values. so it basically compares the bits by bits and if they have compliment of each other then it assigns 1.

In last I'm gonna count the ones and then will weight those strings to make decision. If weighted ones are more then I'll enable the D as a output.

So In this way I'm gonna use fitness function and scoring mechanism for GA.

4) Memory 2 IPD: The memory 0 IPD doesn't store any of the previous moves but memory 2 IPD stores the moves by player. memory 2 IPD acts differently than the memory 0 IPD. They don't act as a same. The memory 2 IPD relies on strategies, whichever strategy we follow, the answer depends on that. Suppose we have CC in memory 2 and we are using tit for tat startegy where in the first iteration the output will always be C. so it acted as per the memory values and startegy used. Moreover, memory 0 IPD doesn't have option to lookup into the memory because they don't keep track of the previous moves. so the output solely depends on the strategy or decision making by player. whereas in memory 2 it's even dependent on the stored memory values.

II. Applied Genetic algorithm for memory 2 IPD

A. Algorithm

```
Data: Memory 2 IPD
Result: optimized strategy
Initialization;
Randomly initialize the population using the range
  with tolerance parameter Decode and calculate the
  function values and transformed values;
while generationNum is less than n do
    Cycle crossover;
    random mutate;
    select best chromosomes;
    if Elitism then
        filter best n members and using elitism keep
          them;
        return elite;
    else
        repeat the process till the best strategy found;
    end
end
```

Algorithm 1: IPD Mem-2 myGA algorithm

B. Matrix of Earnings

Getting the matrix of gains will be done by Game class of my code. Game object takes input parameters and scores them and stores them appropriately.

Here we areinterested in 'No regret' situation of Nash equilibrium. I noted the player 1's move to the player 2's choices and thus I could easily figure out the player 2's best answers to the player 1's strategy. if this has the two max then it's nash equilibrium.

C. Nash Equilibrium

The simultaneous game, where now I know what the other player played so I don't have any regret. The situation where none of the participants can gain by changing the unilateral startegy, while the startegy of other is unchanged.

here we are balacing the score or tradeoff of two outcomes. greedy approach doesn't work all the time when it comes to non-cooperative games.

- (3,3),(0,5),(5,0),(1,1) - 1 equilibrium
- (3,2),(1,1),(0,0),(2,3)- 2 equilibrium
- (1,-1),(-1,1),(-1,1),(1,-1)- 0 equilibrium

D. Pareto Optimal outcome

No situation is superior for both the players thus they don't collectively change the strategy. The superior situation is the situation where both the prisoners are in gain but in this case no situation is superior for any of the prisoner so change in startegy won't make big difference. so in this case it's better to make one side better whithout making the other side worst so pareto optimality is to make something better without harming the another one.

E. Experiment

Creating the random number of game matrix. Randomly enumerate on all generated games and report the nash eqilibrium criteria score. Printing 10 random games found on total of 256 games

[(2, 1), (1, 2), (1, 1), (1, 1)] [(1, 1), (1, 1), (1, 1), (1, 1)] [(1, 2), (2, 1), (2, 2), (1, 2)] [(2, 2), (1, 2), (1, 1), (1, 2)] [(1, 2), (1, 2), (2, 1), (1, 1)] [(2, 1), (2, 1), (1, 1), (1, 1)] [(2, 2), (2, 1), (1, 1), (1, 2)] [(2, 2), (2, 1), (2, 2), (2, 1)] [(2, 2), (2, 1), (1, 2), (1, 2)] [(2, 2), (2, 1), (1, 1), (1, 2)]

Nash Equilibrium: [0: 2, 1: 44, 2: 114, 3: 80, 4: 16]

Incorporating special constraints: After this look for two-shot games whose values are taken in (1,2) and which have exactly the same nash and pareto equilibria and more next to identify the strategy.

F. Strategy

Defining the singular strategy constructors,

```
class Strategy():
    def setMemory(self,mem):
        pass

    def getAction(self,tick):
        pass

    def __copy__(self):
        pass

    def update(self,x,y):
        pass
```

Defining the Periodic Memory based strategy constructors,

```
class Periodic(Strategy):
    def __init__(self, sequence, name=None):
        super().__init__()
        self.sequence = sequence.upper()
        self.step = 0
        self.name = "per_"+sequence
        if (name == None) else name

    def getAction(self, tick):
        return self.sequence[tick
                % len(self.sequence)]

    def clone(self):
        object = Periodic(self.sequence)
        return object

    def update(self,x,y):
        pass
```

Periodic(abc) - A B C A B C A B C A
What if two or more strategy meets? what happen to GA score?
for that I did code different functions such as,
1. meeting - merging to strategies and params
2. reinit - reinitializing the strategy matrix
3. run - for running several rounds of function for one or more strategies. Scoring was done based on best weighted strategies.

G. Tournament

Tournament applies on the set of strategies. Meeting multiple strategies and including them against other strategies is the objective of the tournaments.The score of each strategy is the sum of the scores it has obtained. The winning strategy is the one that gets the highest score. I did create the Tournament function and run function for all these implementation.
Tournament fn- each tournament will be played for all the games and inclusive strategies. Moreover we can create the bag of memory and then again we can play whole tournament.

1) Generating multiple strategies in tournament: Using periodics, generate the multiple strategies and get it ready for ecological competition.

H. Ecological Competitions

The main underlying concept is to consider the representatives of each of the evaluated strategies. The representatives are obtained in proportion with their success in the previous step. You will have more decendents according to your better previous record (fitness).

During these competition, each of the startegy and game is played concurrently with one another. Moreover, the stratas will be merged for getting the performance measurement/comparative analysis.

III. Reactive Strategies

The two main type of strategies are discussed.
tit for tat (TFT)

```
class Tft(Strategy):
    def __init__(self):
        super().__init__()
        self.name = "tft"
        self.hisPast=""

    def getAction(self, tick):
        return 'C' if (tick==0)
                    else self.hisPast[-1]

    def clone(self):
        return Tft()

    def update(self,my,his):
        self.hisPast+=his
```

Spiteful strategy:

```
class Spiteful(Strategy):
    def __init__(self):
        super().__init__()
        self.name = "spiteful"
        self.hisPast=""
        self.myPast=""

    def getAction(self, tick):
        if (tick==0):
                return 'C'
        if (self.hisPast[-1]=='D'
           or self.myPast[-1]=='D') :
            return 'D'
        else :
            return 'C'

    def clone(self):
        return Spiteful()

    def update(self,my,his):
        self.myPast+=my
        self.hisPast+=his
```

Now let's do some performance analysis for the given games using the strategies developed above.

IV. Memory Family

For developing strategic memory family of given dimension, I created one function. It's presented below.

```
class Mem(Strategy):
    def __init__(self, x, y, genome, name=None):
        self.name = name
        self.x = x
        self.y = y
        self.genome = genome
        if (name == None):
            self.name = genome
        self.myMoves = []
        self.itsMoves = []

    def clone(self):
        return Mem(self.x, self.y, self.genome)
```

```
def getAction(self, tick):
    if (tick < max(self.x, self.y)):
        return self.genome[tick]
    cpt = 0
    for i in range(self.x-1,-1,-1):
        cpt*=2
        if (self.myMoves[i] == 'D'):
            cpt+=1
    for i in range(self.y-1,-1,-1):
        cpt*=2
        if (self.itsMoves[i] == 'D'):
            cpt+=1
    cpt += max(self.x, self.y)
    return self.genome[cpt]

def update(self, myMove, itsMove):
    if (self.x > 0):
        if(len(self.myMoves) == self.x):
            del self.myMoves[0]
        self.myMoves.append(myMove)
    if (self.y > 0):
        if(len(self.itsMoves) == self.y):
            del self.itsMoves[0]
        self.itsMoves.append(itsMove)
```

A. Find Best memory(strategy)

Generate the random genotype first.
$def generate_random_genotype(self, x, y)$:

After genrating the random genotype use the selection procedure for selecting the best one. We have seen the overall process that I used to code up the IPD. Now let's see the applied GA process in detail.

V. Applied GA Process

- Encoding
- Fitness function
- Selection
- Crossover and mutation
- Generate new population

A. Encoding the genes

The encoding scheme is pretty simple. 2 bit encoding, 0 for C and 1 for D. As the memory size increases we need more bits to encode. but the representation will be same. 0 for C and 1 for D.
examples:
C - 0
CD - 01
CDDDC - 01110
CDCDCDDC - 01010110

In this way the encoding was done for IPD.

B. Fitness function

Fitness functin I used is scoring of the best individuals. Sorting the individuals on their best scores. Scores are being calculated by,
$scores = Defects * 100 * n - s$
where n is number of opponents and s is previous score.
The scores are sorted in higher to lower order. Thus we can easily take the highest order strategies as a best performance index.

C. Selection

Selection of the best fit strategies are done based on overall weighted average score. The highest weighted score is the best peformative and thus have higher chance of selected as a best strategy. Moreover, the weights depend upon the overall performance of the algorithm in each generation. The number of generation and population scores are important factor for making the decision.

D. crossover and mutation

The crossover probability is generally between 0.5 to 0.7. I chose 0.6 and the reason for that is the trial and run. I was getting more accurate and fast answer for crossProb = 0.6. I did run my algorithm several times with different crossover values.
The selection of mutation probability is important. Initially I kept mutation probability as 0.1. It was not that high but for that value I was getting less accurate xbest. After setting mutation probability to 0.01, I got better xbest. so I did set the mutation probability 0.001 (even lower).

VI. Other Strategies

1) Always Cooperate : Cooperates on every move
2) Always Defect : Defects on every move
3) Tit for Tat : Cooperates on the first move, then simply copies the opponent's last move.
3) Pavlov : Cooperates on the first move, and defects only if both the players did not agree on the previous move.
4) Spiteful : Cooperates, until the opponent defects, and thereafter always defects.
5) Random Player : Makes a random move.
6) Periodic player CD : Plays C, D periodically.
7) Periodic player DDC : Plays D, D, C periodically.
8) Periodic player CCD : Plays C, C, D periodically.
9) Tit for Two Tats : Cooperates on the first move, and defects only when the opponent defects two times.
10) Hard Tit for Tat : Cooperates on the first move, and defects if the opponent has defects on any of the previous three moves, else cooperates.

A. Achieving Nash equilibrium

Dilemma of the prisoner: 1 equilibrium
dip =[(3,3),(0,5),(5,0),(1,1)]
$g = Game(dip, ['C', 'D'])$

The Nash Equilibrium found is,
$\begin{bmatrix}(1,1)\end{bmatrix}$
The random game matrix scores,
$$\begin{bmatrix}(1,4) & (2,4) & (0,4) \\ (2,3) & (1,2) & (2,2) \\ (0,2) & (2,0) & (4,0)\end{bmatrix}$$
The nash equilibrum is, $\begin{bmatrix}(2,4) & (2,3)\end{bmatrix}$

B. Total 256 different Game Generations

Selecting 10 games,
$$\begin{bmatrix}(1,1) & (2,1) & (2,1) & (2,1) \\ (1,2) & (1,2) & (2,2) & (2,1) \\ (2,1) & (2,2) & (1,1) & (1,2) \\ (1,2) & (2,1) & (2,1) & (1,1) \\ (2,1) & (1,1) & (2,2) & (2,1) \\ (1,2) & (1,1) & (2,1) & (1,1) \\ (2,1) & (1,2) & (2,1) & (2,1) \\ (1,1) & (2,1) & (1,2) & (1,1) \\ (1,1) & (2,1) & (1,1) & (1,2) \\ (1,2) & (1,1) & (2,1) & (2,2)\end{bmatrix}$$
Nash equilibrium is, 0: 2, 1: 44, 2: 114, 3: 80, 4: 16

C. Meeting Strategies

Now I did code for the comninational meeting of two strategies, let say s1 and s2 as mentioned below,

```
 # Dilemma of the prisoner
dip =[(3,3),(0,5),(5,0),(1,1)]
g = Game(dip,['C','D'])
s1=Periodic("CCD")
s2=Periodic("DDC")
m = Meeting(g,s1,s2,10)
m.run()
```

Results are:
$periodic - CCDCCDCCDCCDC$ 15
$periodic - DDCDDCDDCDDCD$ 35

D. Playing tournaments

Playing the tournament between,
$periodic - C(allCs)$,
$periodic - D(allDs)$,
$periodic - DDC$,
$periodic - CCD$,

Results of tournament of 4 strategies					
Strategy	perD	perDDC	perCCD	perC	Total
perD	10	22	38	50	120
perDDC	7	16	35	44	102
perCCD	3	15	24	36	78
perC	0	9	21	30	60

1) Analysis:

1) Analysis: periodic D startegy has the highest fitness score. so overall for all the strategy meetings in tournament following the periodic D can earn the highest scores thus more preferreble. defectig everytime can be better strategy in meetings of strategies in tournament. moreover always defecting is also quite good as per performance score. It's rare but still helpful. Periodic CCD is also better.

E. Ecological competitions

Results of startegies in 10 iterations						
Generation	perD	perDD	perDDD	perCDD	perDCD	perDDC
0	100	100	100	100	100	100
1	133	133	133	111	111	111
2	171	171	171	117	118	118
3	213	213	213	118	119	119
4	256	256	256	113	114	114
5	298	298	298	102	103	103
6	336	336	336	87	88	88
7	369	369	369	71	72	72
8	396	396	396	55	56	56
9	417	417	417	41	41	41
10	433	433	433	29	29	29

Note: I did this till the 20 iterations and more results are uploaded in Jupyter notebook PDF.

Results of startegies in 10 iterations								
Gen	perCD	perDC	perCCD	perCDC	perDCC	perC	perCC	perCCC
0	100	100	100	100	100	100	100	100
1	100	100	88	88	88	66	66	66
2	94	94	73	73	73	40	40	40
3	84	84	56	56	56	21	21	21
4	70	70	39	39	39	9	9	9
5	54	54	25	25	25	3	3	3
6	39	39	14	14	14	0	0	0
7	26	26	7	7	7	0	0	0
8	16	16	3	3	3	0	0	0
9	9	9	1	1	1	0	0	0
10	5	5	0	0	0	0	0	0

Note: I did this till the 20 iterations and more results are uploaded in Jupyter notebook PDF.

1) Analysis: Here also I'm getting the same result. As not everytime defecting helps but it helps in drastic way. all Ds have higher score then everything else. By judging both the tables we can clearly see that, finding tradeoff between always cooperating and defecting can be in our favour, the same as tit for tat with forgiveness so this proves that.

F. Tournament on Reactive strategies

The 4 main strategies are taken into account, 1. Tit for tat
2. allC
3. allD
4. periodicCCD

Results of startegies in 10 iterations				
i	tft	perC	perCCD	perD
0	100	100	100	100
1	103	85	96	114
2	108	72	91	127
3	114	60	85	138
4	123	50	79	146
5	135	42	73	148
6	150	36	68	144
7	169	31	64	133
8	1192	28	61	116
9	219	26	58	94
10	247	25	56	70

Note: I did this till the 20 iterations and more results are uploaded in Jupyter notebook PDF.

1) Analysis: More detailed analytical proofs are presented below

The complete matrix					
Strategy	perD	tft	perCCD	perC	Total
perD	1000	1004	3668	5000	10672
tft	999	3000	2667	3000	9666
perCCD	333	2667	2334	3666	9000
perC	0	3000	2001	3000	8001

The Winners of the tournament	
Strategy	Total
perD	10672
tft	9666
perCCD	9000
perC	8001

Final Populations ranked	
Strategy	Total
tft	390
perC	9
perCCD	0
perD	0

G. Memory Family

Running myGA to the various strategies and ranking the strategies based upon the best ranked strategies. Finding answer for the generalized best strategy.
I'm using TFT(tit for tat), spiteful, allD, allC, per-CD, per-DC.

Results analysis of 7 strategies							
Strategy	tft	perC	perD	perCD	perDC	spiteful	Total
spiteful	300	300	99	297	299	300	1595
tft	300	300	99	248	250	300	1497
perD	104	500	100	300	300	104	1408
perCD	253	400	50	200	250	57	1210
perDC	250	400	50	250	200	54	1204
perC	300	300	0	150	150	300	1200

VII. Conclusion

In conclusion, The GA can be applied efficiently to solve the IPD kind of Game theory problems. The strategies highly depends on situation. No free lunch! You need to set up the parameters and then judge each strategy independently for getting best outcome in your favor.

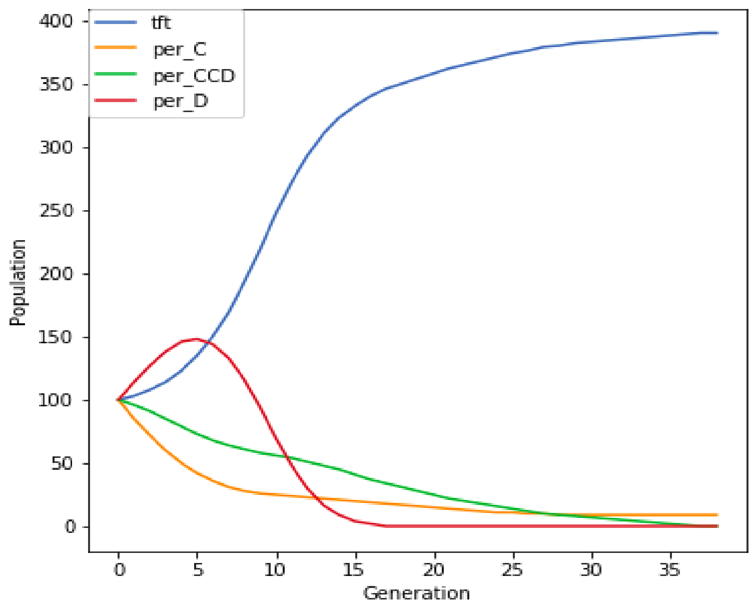

Fig. 2: Generations vs population performance graph for given strategies

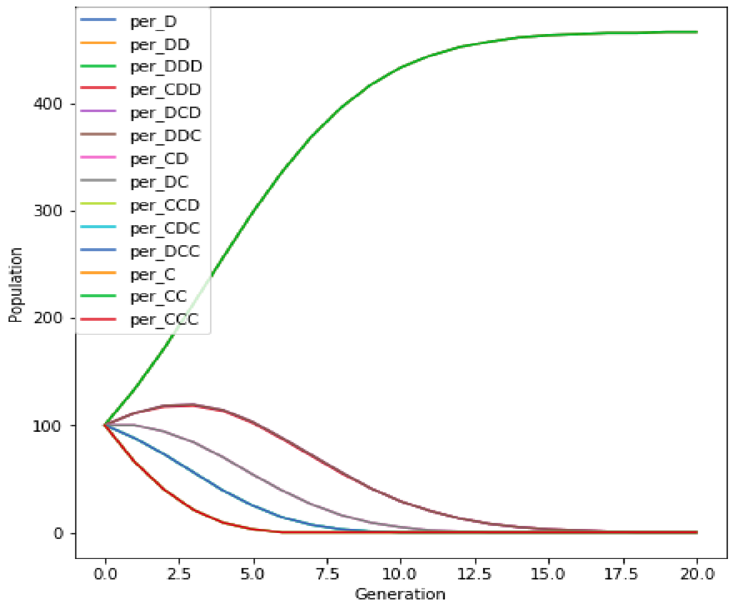

Fig. 3: Generations vs population performance graph for given strategies

References

[1] Robert Axelrod, The Evolution of Cooperation (New York: Basic Books, 1984).

[2] JP Delahaye and P Mathieu. Surprises in the world of cooperation. For Science, special issue "Social Mathematics", pp 58-66, July 1999.

[3] Philippe Mathieu, Jean-Paul Delahaye. [New Winning Strategies for the Iterated Prisoner's Dilemma] (http://jasss.soc.surrey.ac.uk/20/4/12.html).

J. Artificial Societies and Social Simulation 20 (4) (2017)

[4] Bruno Beaufils, Jean-Paul Delahaye and Philippe Mathieu. Our Meeting with Gradual: A Good Strategy for the Itareted Prisoner ’s Dilemma, in Intern. Cof. on Artificial Life V (ALIFE V), pp. 159-165, 16-18 May 1996, Nara, Japan.

[5] Philippe Mathieu, Jean-Paul Delahaye. New Winning Strategies for the Iterated Prisoner’s Dilemma. AAMAS 2015: 1665-1666