UNDERGRADUATE RESEARCH IN NATURAL AND CLINICAL SCIENCE AND TECHNOLOGY (URNCST) JOURNAL
Read more URNCST Journal articles and submit your own today at: https://www.urncst.com

REVIEW **OPEN ACCESS**

Review: A Genetic Algorithm Approach to the Iterated Prisoner's Dilemma

David R. Collins, BCS Student [1]*

[1] Department of Computer Science, University of Windsor, Windsor, Ontario, Canada N9B 3P4

*Corresponding Author: colli11j@uwindsor.ca

Abstract
Introduction: The use of machine learning tactics such as the Moran process and evolutionary finite state machines have the potential to outperform classic strategies for the iterated prisoner's dilemma.
Methods: Applying a genetic algorithm approach to the iterated prisoner's dilemma while modeling strategies with finite state machines proved to be an efficient method in which the produced strategies were able to cooperate unilaterally with their opponents.
Results: Varying parameters in the evolution process such as the amount of generations, population size and bottleneck size were shown to directly contribute to the success of the strategies produced. In comparing various optimization methods, the genetic algorithm utilizing finite state machines outperformed the Moran process with respect to the highest scoring strategies produced by each.
Conclusions: Tangible tactics can be extracted from these strategies, which were evolved using standard genetic algorithm tactics utilizing gene crossover and mutation techniques.

Keywords: genetic algorithm; artificial intelligence; machine learning; iterated prisoner's dilemma; finite state machine; Moran process

Introduction

The Iterated Prisoner's Dilemma (IPD) is a popular topic of scholarly research in the fields of game theory, psychology of cooperative behavior, and artificial intelligence. The iterated prisoner's dilemma is an extension of the original single round game scenario proposed in 1950 by Merrill Flood and Melvin Dresher while working at RAND [1]. The difference with the iterative version is that the two participants have the opportunity to learn and make decisions following the behavioral tendencies of their opponent [2].

The game can be presented with the scenario of two criminals being questioned for a crime separately. Each criminal is offered a deal involving a lesser punishment for betraying their partner in crime. If both criminals decide to betray each other, they receive a punishment of ten years in prison. If neither of them betrays (meaning they cooperate with each other) then each receives a small charge of one year. The problem gets interesting in the last scenario, where one prisoner decides to betray, while the other chooses to cooperate. In this case, the betrayer gets away free, while the cooperative criminal now must serve a full sentence of twenty years [1].

The paradox of the iterated prisoner's dilemma is that if both parties just act in their best personal interest, it will not result in the most optimal outcome [2]. Hence, since both players can expect to interact again after the round finishes, they must make a rational decision that will influence their opponent towards highest mutual gain.

Interest in developing strategies towards playing the iterated prisoner's dilemma came from a researcher named Robert Axelrod, who proposed a tournament where competitors from around the world applied their computer programmed strategies against each other's [3]. The winning algorithm in this first tournament was Tit-For-Tat, a deterministic strategy that always cooperated on the first move, then chose to repeat the opponents last move for the rest of the game [4]. From Axelrod's experiments and research, he concluded that in order for a strategy to be successful, it must be nice (optimistically cooperate), know when to retaliate (sometimes you must show the opponent the consequences of betrayal), know when to forgive (not retaliate when it will just lead to revenge and counter-revenge), and be non-envious (not trying to outscore the opponent) [2]. However, the point of this research paper is not to create our own optimal strategy, rather study the outcome of applying machine learning.

Genetic algorithms are one particular field of many machine learning models that apply well to the iterated prisoner's dilemma [5]. Genetic algorithms involve the process of natural selection to generate solutions for optimization and search problems [5]. This is explained in detail with the following sections of this paper, but a high-level view of using a genetic algorithm to optimize the prisoner's dilemma involves encoding the strategy using various methods such as bit string, or finite state machines. The encoded strategy can be viewed as a chromosome from a biological perspective, which will undergo mutation and reproduction in order to generate the highest performance. Finite state machines will be explained in further detail, but briefly put: they provide a specific action to perform given the current state in a problem. Finite state machines can be very powerful in their ability to facilitate reproduction and evolve [6].

A well-documented python-based library named Axelrod-Dojo was used to simulate IPD tournaments and extract results [7].

The motivation behind applying a genetic algorithm approach to the iterated prisoner's dilemma is the desire to discover optimal choices for mutation rates, population sizes, memory depths (the amount of previous turns to base the current choice off of), as well as extracting some useful strategies from the machine-generated outcomes of multiple experiments.

Methods

The Moran Process

The Moran Process is a widely used population model resembling the actions of natural selection [8]. The way this model works is it starts with a fixed population size of N individuals who interact with each other over many rounds. When the players interact with each other every round, a score is given to both. In the case of the iterated prisoner's dilemma, the game never changes and has very simple rules (previously described above). Each individual's allocated score will influence their fitness levels, which in turn affects their chances of reproduction. Fitness proportionate selection is a very important factor in many genetic algorithm models, including the Moran process [8]. In the case of the iterated prisoner's dilemma, many rounds can be played between two opponents allowing their fitness score to represent the sum of points gained through a series of iterated rounds.

In the implementation of the Moran process within the Axelrod library, there is a random chance that one player will be replaced by another. Other factors can contribute to the reproductive process, such as mutation rate [5]. Having mutations within a genetic model such as the Moran process is extremely beneficial to ensuring genetic diversity within populations [8]. Genetic diversity is a biological factor that ensures evolving populations carry unique genetic information (or encoded strategies, in the case of IPD) [5]. Mutation alters usually one gene at random allowing the chromosomes in a population stay different from each other [5]. It is important to note that the mutation rate in a genetic model should remain low rather than being high. If mutation rates are too high, the search technique ends up resembling a random search, since reproductive measures won't be able to carry much effect. Restated, high mutation rates have the chance of reversing or scrambling the progress made in the right direction by combining the best aspects of individuals in a population (via reproduction).

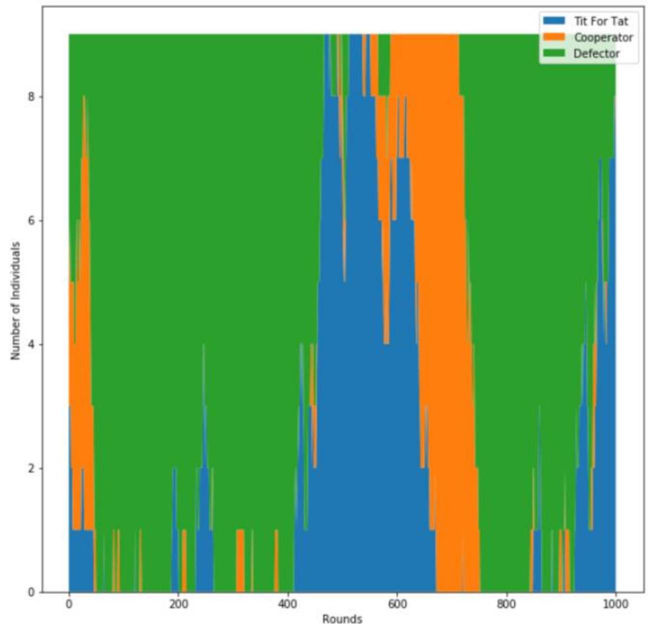


Figure 1: Dynamic population levels of strategies using the Moran Process.

In context of machine learning and search algorithms, a common problem that occurs is the searching technique getting stuck in a local minima [5]. If for example reproduction was converging in a negative direction, without mutation the search state may never recover from this minimum position. As one can see, having the right mutation rate can be crucial for a reproductive model.

Coming back to the efforts of the Moran process, the events explained continue over many rounds until the population consists of a single individual. This phenomenon is known as fixation [8]. It is important to note that when mutation is implemented in the Moran process, there is a possibility that fixation may never occur, but the population may heavily favor one individual as seen in Figure 1. This can easily be solved by fixing the number of rounds for the population to reproduce. After one has an understanding of the Moran process, the observation can be made that this search technique is closely related to another technique called Beam search. The reason the Moran process is similar to Beam search is because in the Moran process recombination does not actually involve crossing over of alleles, rather the same strategies exist within the population at varying levels [8]. Beam search works in a similar way, in that the highest performing "branches" of each iteration / round of search are used to continue search from, resulting in the population tending towards fixation of a strategy with the highest fitness level [9]. The real genetic recombination happens in the method explained next, finite state machines.

Finite State Machines

Finite state machines are a system where particular inputs cause an action to occur depending on the current "state" thus far [6]. The state is often viewed as a node, while edges to other states represent the action to take to move to a new state. These mappings from state/action pairs to other states allows for specific instructions to be encoded with ease. It is important to note that a finite state machine always requires a starting state [6]. With respect to the iterated prisoner's dilemma, finite state machine can be used to represent a given strategy with only tuples in this form: (current state, the opponent's latest action, the state to move to, the action to take).

For example, the well-known IPD strategy "Tit-for-Tat". This strategy initially cooperates, then replicates the opponents move for the rest of the game [4]. A finite state machine to represent this strategy shown in Figure 2 is as follows: (1, C, 1, C) (1, D, 1, D) with initial state 1, and initial move cooperate. This roughly translates to "if in state 1, the opponent cooperates, continue to state 1 and cooperate. If they defect, also continue to state 1 and defect."

Although this finite state machine only has one state, there can be many states depending on the strategy. Finite state machines as strategies for the IPD will end up basing their moves of the opponent's history of moves. This is similar to the bit-string encoded method described earlier, but slightly more complex in that instead of having a fixed history of opponent moves to decide which action to take (like the bit string encoded method also known as lookup-tables), the finite state machine will make its move depending on which state its currently in. One of the main advantages to using finite state machines instead of lookup tables is that many lookup table rows and their actions are redundant; they could easily be represented by a finite state machine while using less space to encode the states and actions.

Population genetics as a whole have served as a major influence in genetic algorithm models, and have proven to give great results in search spaces involving optimization problems (such as the iterated prisoner's dilemma) [5]. The results of this application can be seen in the following sections of this article.

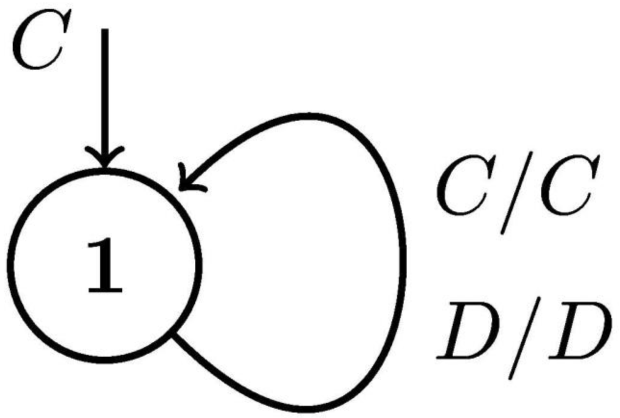


Figure 2: Finite state machine representing the "Tit-for-Tat" strategy.

Experimental Setup

In setting up this experiment, the main programming language used is Python version 3. The open source library used is the Python Axelrod library, which was developed to test and implement algorithms for the iterated prisoner's dilemma [7]. As noted, the effort of this paper was directed towards researching IPD, not developing a game-suite to run IPD algorithm simulations, which is why the Axelrod library was chosen to build upon. In the Axelrod library, many of the classic algorithms, such as Tit-for-Tat, defector, cooperator, etc. are already implemented and ready for use [7]. One main approach was to use the built in Moran process abilities of the library to experiment with changing population sizes, starting populations, mutation rates and round sizes.

The first experiment involved using a starting population of nine individuals. There were three individuals per strategy, with the starting IPD strategies being Tit-for-tat, defector and cooperator. Here are the brief explanations of each strategy:

Tit-for-tat: This strategy always cooperates on the first move, then plays whatever its opponent did on the last turn [4].

Defector: always chooses to defect instead of cooperate [1]. It is important to note that from a purely individual standpoint, this strategy theoretically has the highest yield of points. As explained previously, the paradox of the IPD is that in order to achieve success, the individual must act rationally and cooperate with the opponent.

Cooperator: always cooperates, regardless of opponents move. Essentially the opposite of defector [1].

The starting population of 9 individuals then played each other using the standard scoring and rules of the iterated prisoner's dilemma. The games were fixed at a length of 200 turns in order to ensure a large enough representation of the two opponents' strategy against each other. After completing their matches, the population would undergo reproduction and mutation. The mutation rate of this experiment was fixed at 5%. This sequence of events represents one round. Conducting this experiment involved completing 1000 rounds, measuring the distribution of the population each round.

Since the Moran process is a genetic model, each strategy's population levels vary from round to round, and the strategy that carried the highest population levels each round is recorded and graphed for visualization [8]. By the end of the 1000 rounds, it was observed that Tit-for-Tat was the winner since it well over the majority of the population implemented this strategy.

Although the Moran process approach is very interesting and serves as a great first example for optimizing the iterated prisoner's dilemma, finite state machines allow the strategies to completely evolve through reproduction and mutation, creating machine-generated strategies for the IPD [6].

The genetic algorithm used to optimize the search space while implementing populations of finite state machine players has three major parameters: the population size, the bottleneck parameter and the mutation probability [7]. The importance and effect of the mutation probability has already been discussed. The population size corresponds to the number of individuals participating from round to round. The bottleneck is a very important parameter, which indicates how many of the strategies ranked via the fitness function best to worst, will move on to the next generation. Biological systems naturally have a bottleneck on their populations, which is why genetic algorithms have chosen to implement this evolution-inspired factor [5]. In general, the genetic algorithms goal is to use key systems found in the evolution process in order to produce the best candidate solutions to a problem [2].

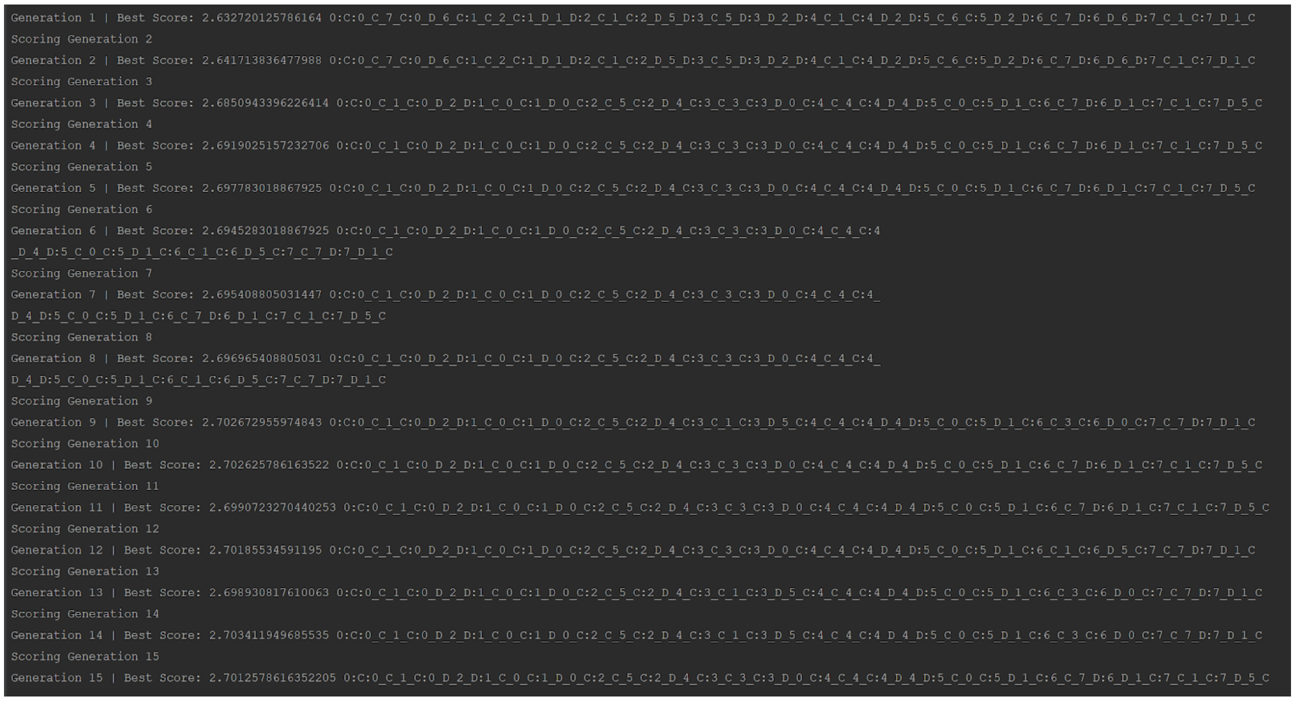
Using this genetic algorithm approach, the finite state machines will undergo reproduction and mutation at the end of each round while competing against the other strategies for the iterated prisoner's dilemma. Understanding how these state machines create new strategies via crossover and mutation of their alleles is very important. To undergo crossover, a randomly selected number of states/action pairs from a finite state machines encoded strategy are switched with another finite state machines state/action pairs. This process simulates genetic recombination (gene crossover of meiosis) in biological systems [5].

Mutation of the finite state machine strategies occurs with a carefully selected probability (choice of value discussed later) in which one of the state/action pairs will be switched to another state/action pair within the strategy. There is also a mutation probability that the initial action and initial state of the finite state machine will be switched to a random action/state of the finite state machine.

With this method of reproduction described, the results of performing this genetic algorithm will be described next.

Results

With the objective function set to score, the finite state machine strategies used their summed score as a fitness measure while reproducing. The first attempt to produce a high performing IPD finite state machine involved running the genetic algorithm with the following parameters: the bottleneck, which determines how many species of the population to allow to continue to the next generation was set to 10, the amount of generations to run the genetic algorithm over was set to 15, mutation rate set to 0.1 (or 10% of the time a state/action pair would randomly switch to another state/action pair within that FSM, and other mutations may occur like the start state or start action changing), the objective function (also known as the fitness function which evaluates the performance of each strategy) was set to the average score of the turns in a single round, the population was set to 15 (meaning that the strategy started with 15 random preset strategies such as TFT, TF2T, Defector etc.) and stayed at this level throughout the process, each round was set to 100 turns and each round was played over 3 repetitions [10]. Another important factor about this first experiment was the total allowed states to be used was fixed at 8, meaning that strategies couldn't just create an infinite amount of states. The reason having the amount of states fixed at a small number is important is because without this, the genetic algorithm would essentially be pattern matching [9]. This means that in practice, the genetic algorithm could create a depth of 100 states, one per turn, and create a pattern to correctly maximize every opponent in the population (which is clearly NOT the desired result of applying the genetic algorithm).



```
Generation 1 | Best Score: 2.632720125786164 0:C:0_C_7_C:0_D_6_C:1_C_2_C:1_D_1_D:2_C_1_C:2_D_5_D:3_C_5_D:3_D_2_D:4_C_1_C:4_D_2_D:5_C_6_C:5_D_2_D:6_C_7_D:6_D_6_D:7_C_1_C:7_D_1_C
Scoring Generation 2
Generation 2 | Best Score: 2.641713836477988 0:C:0_C_7_C:0_D_6_C:1_C_2_C:1_D_1_D:2_C_1_C:2_D_5_D:3_C_5_D:3_D_2_D:4_C_1_C:4_D_2_D:5_C_6_C:5_D_2_D:6_C_7_D:6_D_6_D:7_C_1_C:7_D_1_C
Scoring Generation 3
Generation 3 | Best Score: 2.6850943396226414 0:C:0_C_1_C:0_D_2_D:1_C_0_C:1_D_0_C:2_C_5_C:2_D_4_C:3_C_3_C:3_D_0_C:4_C_4_C:4_D_4_D:5_C_0_C:5_D_1_C:6_C_7_D:6_D_1_C:7_C_1_C:7_D_5_C
Scoring Generation 4
Generation 4 | Best Score: 2.6919025157232706 0:C:0_C_1_C:0_D_2_D:1_C_0_C:1_D_0_C:2_C_5_C:2_D_4_C:3_C_3_C:3_D_0_C:4_C_4_C:4_D_4_D:5_C_0_C:5_D_1_C:6_C_7_D:6_D_1_C:7_C_1_C:7_D_5_C
Scoring Generation 5
Generation 5 | Best Score: 2.697783018867925 0:C:0_C_1_C:0_D_2_D:1_C_0_C:1_D_0_C:2_C_5_C:2_D_4_C:3_C_3_C:3_D_0_C:4_C_4_C:4_D_4_D:5_C_0_C:5_D_1_C:6_C_7_D:6_D_1_C:7_C_1_C:7_D_5_C
Scoring Generation 6
Generation 6 | Best Score: 2.6945283018867925 0:C:0_C_1_C:0_D_2_D:1_C_0_C:1_D_0_C:2_C_5_C:2_D_4_C:3_C_3_C:3_D_0_C:4_C_4_C:4
_D_4_D:5_C_0_C:5_D_1_C:6_C_1_C:6_D_5_C:7_C_7_D:7_D_1_C
Scoring Generation 7
Generation 7 | Best Score: 2.695408805031447 0:C:0_C_1_C:0_D_2_D:1_C_0_C:1_D_0_C:2_C_5_C:2_D_4_C:3_C_3_C:3_D_0_C:4_C_4_C:4_
D_4_D:5_C_0_C:5_D_1_C:6_C_7_D:6_D_1_C:7_C_1_C:7_D_5_C
Scoring Generation 8
Generation 8 | Best Score: 2.696965408805031 0:C:0_C_1_C:0_D_2_D:1_C_0_C:1_D_0_C:2_C_5_C:2_D_4_C:3_C_3_C:3_D_0_C:4_C_4_C:4_
D_4_D:5_C_0_C:5_D_1_C:6_C_1_C:6_D_5_C:7_C_7_D:7_D_1_C
Scoring Generation 9
Generation 9 | Best Score: 2.702672955974843 0:C:0_C_1_C:0_D_2_D:1_C_0_C:1_D_0_C:2_C_5_C:2_D_4_C:3_C_1_C:3_D_5_C:4_C_4_C:4_D_4_D:5_C_0_C:5_D_1_C:6_C_3_C:6_D_0_C:7_C_7_D:7_D_1_C
Scoring Generation 10
Generation 10 | Best Score: 2.702625786163522 0:C:0_C_1_C:0_D_2_D:1_C_0_C:1_D_0_C:2_C_5_C:2_D_4_C:3_C_3_C:3_D_0_C:4_C_4_C:4_D_4_D:5_C_0_C:5_D_1_C:6_C_7_D:6_D_1_C:7_C_1_C:7_D_5_C
Scoring Generation 11
Generation 11 | Best Score: 2.6990723270440253 0:C:0_C_1_C:0_D_2_D:1_C_0_C:1_D_0_C:2_C_5_C:2_D_4_C:3_C_3_C:3_D_0_C:4_C_4_C:4_D_4_D:5_C_0_C:5_D_1_C:6_C_7_D:6_D_1_C:7_C_1_C:7_D_5_C
Scoring Generation 12
Generation 12 | Best Score: 2.70185534591195 0:C:0_C_1_C:0_D_2_D:1_C_0_C:1_D_0_C:2_C_5_C:2_D_4_C:3_C_3_C:3_D_0_C:4_C_4_C:4_D_4_D:5_C_0_C:5_D_1_C:6_C_1_C:6_D_5_C:7_C_7_D:7_D_1_C
Scoring Generation 13
Generation 13 | Best Score: 2.698930817610063 0:C:0_C_1_C:0_D_2_D:1_C_0_C:1_D_0_C:2_C_5_C:2_D_4_C:3_C_1_C:3_D_5_C:4_C_4_C:4_D_4_D:5_C_0_C:5_D_1_C:6_C_3_C:6_D_0_C:7_C_7_D:7_D_1_C
Scoring Generation 14
Generation 14 | Best Score: 2.70341194968553 5 0:C:0_C_1_C:0_D_2_D:1_C_0_C:1_D_0_C:2_C_5_C:2_D_4_C:3_C_1_C:3_D_5_C:4_C_4_C:4_D_4_D:5_C_0_C:5_D_1_C:6_C_3_C:6_D_0_C:7_C_7_D:7_D_1_C
Scoring Generation 15
Generation 15 | Best Score: 2.7012578616352205 0:C:0_C_1_C:0_D_2_D:1_C_0_C:1_D_0_C:2_C_5_C:2_D_4_C:3_C_3_C:3_D_0_C:4_C_4_C:4_D_4_D:5_C_0_C:5_D_1_C:6_C_7_D:6_D_1_C:7_C_1_C:7_D_5_C
```

Figure 3: Evolution of FSM over 15 generations showing highest scores and state layout for each generation.

As one can see in Figure 3, each generation's highest scoring strategy is displayed, along with its score (which is an average of its score from each turn played). The highest score of the last generation (Gen. 15) is 2.70. To put this number in perspective, a sample tournament was run using the predefined strategies: Tit-for-tat, Cooperator, Defector, Grudger, and Random. The first three strategies have been explained, and Grudger is a strategy that always cooperates until the opponent plays defect, then holds a "grudge" for the rest of the match, playing defect no matter what [11]. The random strategy chooses to defect or cooperate at random each round. The results of the sample strategy are shown in Figure 4.

The results show defector having the highest score of 2.5, followed by Grudger then Tit for Tat scoring a 2.3. The finite state machine generated in the first experiment yielded a score of 2.7, performing higher than these four popular strategies in the sample tournament.

Here is that finite state machine visually represented in a graph, with the edges between states representing the opponent's action, followed by a slash ('/'), then the action it will take that round while shifting to the new state. 'C' stands for cooperate, and 'D' stands for defect. Notice the starting move is cooperate, and the starting state is labelled as '0'. The disconnected subgraphs of the encoded finite state machine were removed for simplicity sake shown in Figure 5.

One important point to discuss about the finite state machine approach is that these strategies are not "memory one", meaning they don't simply choose their next move depending on the opponents last move (like tit for tat) [4]. In fact, finite state machines don't have a "memory" as lookup tables do, rather the action to take only depends on which state is currently being used [6]. This strategy had a maximum of 8 states, and only ended up using 5 in the final generation. When running the same evolutionary algorithm with a maximum of 3 states instead of 8, the results only decreased to a best score of 2.67, which is only a 1.1% difference.

The small reduction in performance is not as large as one might hypothesize, mostly because having three states still allows for quite complex strategies to exist, even compared to simple strategies like Tit-for-Tat (which is a one state / one memory strategy) [6].

The same experiment was performed again, this time with a smaller population of size 10, a proportionately smaller bottleneck of size 3, same mutation rate of 10%, with 8 states but over 50 generations instead of only 15. The results showed a best score of 2.78, which is 2.9% higher than the last experiments score of 2.70. These results and similar retesting lead to the conclusion that performing the genetic algorithm over a larger number of generations yielded higher results. It is expected that the increased performance comes from more genetic recombination and mutation chances (over the 50 generations).

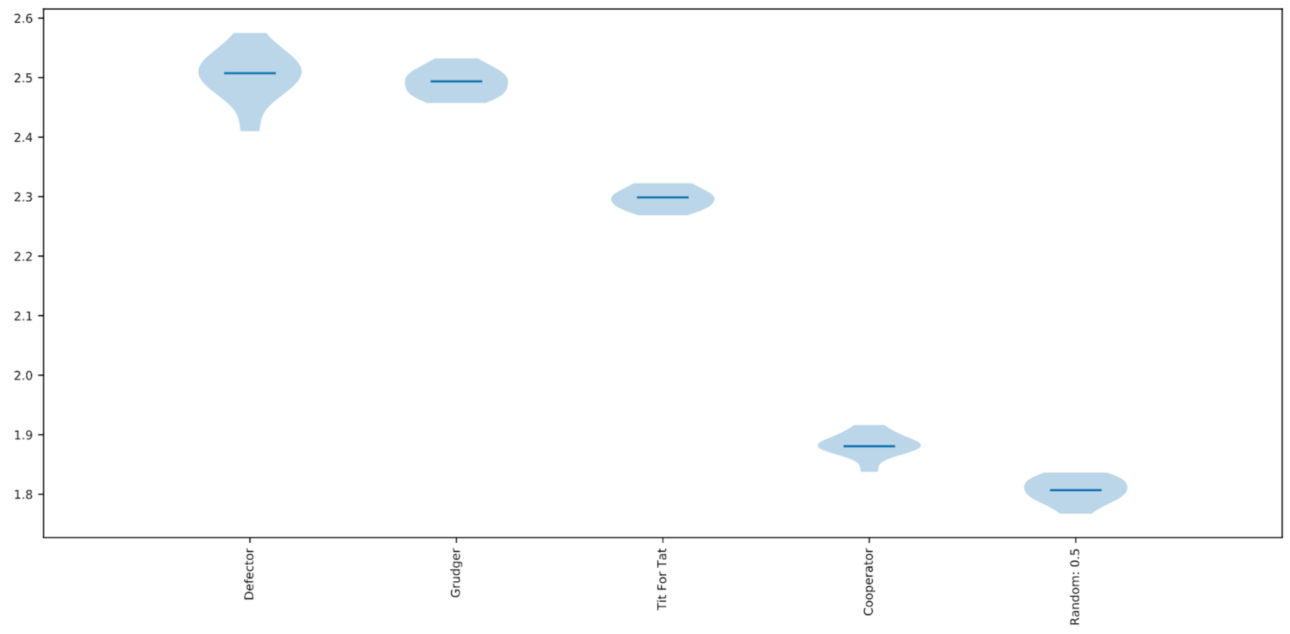


Figure 4: Average fitness scores of stock strategies played against each other in sample tournament

.

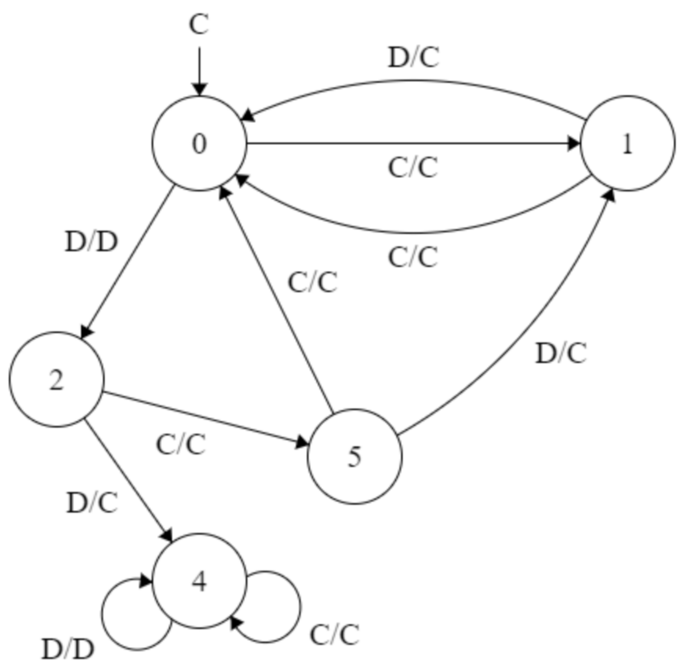


Figure 5: 8-State evolved finite state machine after 15 generations of genetic algorithm.

Recombination creates more opportunity for the finite state machines to evolve and utilize the highest performing mappings from states to actions through the genetic process [5]. Following these results, the same test was performed over 500 generations. The results re-enforced the hypothesis of increasing generations leading to higher performance, with a best score of 2.87. The results are shown in Figure 6.

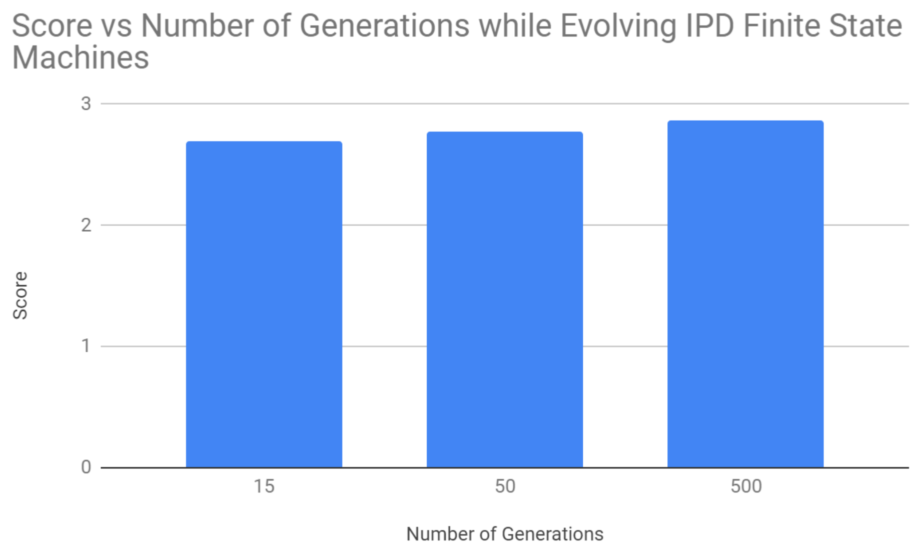


Figure 6: Highest scoring evolved finite state machine strategies varying according to number of generations during genetic process.

Here is the highest scoring 8-state strategy produced through evolution of finite state machine strategy:

Generation 500 | Best Score: 2.870754716981132
0:C:0_C_5_C:0_D_4_D:1_C_0_D:1_D_2_D:2_C_5_C:2_
D_4_D:3_C_0_D:3_D_2_D:4_C_7_C:4_D_7_D:5_C_5_C:
5_D_4_D:6_C_0_D:6_D_2_D:7_C_7_D:7_D_2_C

Some tactics and strategies of the iterated prisoner's dilemma can be extracted from the high scoring finite state machines produced through the genetic algorithm. These tactics reinforce Robert Axelrod's conclusions from his research, namely: strategies must be nice, know when to retaliate, know when to forgive and be non-envious (previously described) [11]. As you can see, this strategy employs some clever tactics, such as choosing to defect even after the opponent cooperates a few times, but not always [2]. This strategy chooses to defect after the opponent cooperates in states 1, 3, 6, and 7 (total of 4 states). The rest of the states show cooperation after cooperation, defection after defections, and some forgiving states of cooperation after defection. It is observed that this strategy is nice and forgiving, but also takes the chance to defect after the opponent cooperates to maximize its scoring potential.

It is also important to note that using a bottleneck on the population contributed to the evolution of new species [10]. Population bottlenecks cause a decrease in genetic variation because infrequently occurring alleles (which are a variant form of a gene) face a greater chance of being excluded from the new generation [10]. This can cause the new generation to carry species that are genetically distinct from the previous, leading to the evolution of new species, and hence new strategies in the context of our genetic algorithm [2].

Discussion

Various genetic models were observed and tested to measure their performance in generating iterated prisoner's dilemma strategies with the results recorded and analyzed. The first genetic model, the Moran process was used to maximize the strategies within the search space of a fixed population of predetermined classical strategies. The results of applying this model showed that natural selection is a powerful and efficient tool when applied to the iterated prisoner's dilemma, as competing strategies lead to the fixation of the population. As predicted, Tit-For-Tat was shown as the highest scoring amongst other simple strategies.

The second approach while applying evolutionary models to the IPD was the use of finite state machines to represent individuals in a population of IPD strategies. Finite state machines composed of state and action pairs representing what move to make depending on the opponents move in game were used to create powerful strategies that outperformed the average scores of the classic IPD strategies [11]. The use of genetic recombination and mutation showed to be very effective in elevating the scores of strategies throughout many generations. The use of a bottleneck to select a proportion of individuals to move onto the next generation contributed to the production of new strategies within the population [10].

The results of evolving the finite state machine provided useful insight on tactics that can be used in all strategies to increase scoring potential. These tactics resembled the factors a successful strategy must have proposed by Robert

Axelrod's studies, such as balancing cooperative efforts with the correct amount of punishment via defection [2].

Conclusions

After observing the performance levels of the evolved finite state machine strategies, it became apparent that genetic algorithms perform well in optimization problems such as the iterated prisoner's dilemma, and contribute strategies that otherwise would not have been formed without the help of machine learning tactics. The paradox of the iterated prisoner's dilemma is that the player cannot act unilaterally in its best interest; it must work with the opponent to maximize its score [1]. The genetic algorithm applications shown in this paper were able to produce strategies that worked around this game-theoretic dilemma and showed results that beat the scores of traditional strategies.

List of Abbreviations

FSM: Finite state machine
IPD: Iterated prisoner's dilemma
GA: Genetic algorithm
TFT: Tit-for-tat

Conflicts of Interest

The author declares that they have no conflicts of interest.

Ethics Approval and/or Participant Consent

This study did not require ethics approval or participant consent as it is a technical review based off of genetic algorithms.

Authors' Contributions

DC: made contributions to the design of the study, collected and analysed data, drafted the manuscript, and gave final approval of the version to be published.

Acknowledgements

This research was originally conducted as a project for an artificial intelligence class taught by Dr. Robin Gras at the University of Windsor.

Funding

This study was not funded.

References

[1] Kuhn S. Prisoner's Dilemma [Internet]. Stanford Encyclopedia of Philosophy. Stanford University; 2014 [cited 2018Mar10]. Available from: https://plato.stanford.edu/archives/spr2017/entries/prisoner-dilemma/
[2] Axelrod RM. The evolution of cooperation. New York: Basic Books; 2006.
[3] Knight, V. Reference — Axelrod 0.0.1 documentation [Internet]. Axelrod.readthedocs.io; 2015 [cited 2018Mar11]. Available from: https://axelrod.readthedocs.io/en/stable/reference/index.html
[4] Only a Game. Tit for Tat. [Internet]. 2017 [cited2018Mar12] Available from: https://onlyagame.typepad.com/only_a_game/2007/06/tit_for_tat.html
[5] Carr, J. An Introduction to Genetic Algorithms. [Internet] Whitman.edu; 2015 [cited 2018Mar10]. Available from: https://www.whitman.edu/Documents/Academics/Mathematics/2014/carrjk.pdf
[6] SZ. Finite State Automata [Internet]. 2019 [cited 2018Mar14]. Available from: https://www.cs.odu.edu/~zeil/cs390/latest/Public/fsa/index.html
[7] Axelrod-Python. Axelrod-Python/axelrod-dojo [Internet]. GitHub; [cited 2018Mar12]. Available from: https://github.com/Axelrod-Python/axelrod-dojo
[8] Moran, P. A. P. Random processes in genetics. Mathematical Proceedings of the Cambridge Philosophical Society; 1958.
[9] Techopedia.com. What is Pattern Matching? - Definition from Techopedia. [Internet]. 2018 [cited 2018Mar14] Available from: https://www.techopedia.com/definition/8801/pattern-matching
[10] Nature.com. population bottleneck | Learn Science at Scitable. [Internet]; 2016 [cited2018Mar10]. Available from: https://www.nature.com/scitable/definition/population-bottleneck-300
[11] Prisoners-dilemma.com. Prisoner's Dilemma. [Internet] [cited 2018Mar12] Available from: http://www.prisoners-dilemma.com/strategies.html

Article Information

Managing Editor: Jeremy Y. Ng
Peer Reviewers: Ikjot Saini, Kalyani Selvarajah, Mahreen Nasir Butt, Pooya Moradian Zadeh
Article Dates: Received Aug 14 19; Published Sep 19 19

Citation
Please cite this article as follows:
Collins D.R. Review: A genetic algorithm approach to the iterated prisoner's dilemma. URNCST Journal. 2019 Sep 19: 3(8). https://urncst.com/index.php/urncst/article/view/155
DOI Link: https://doi.org/10.26685/urncst.155