CT421 Assignment 2

Liam Holland - 21386331

March 2025

1 Iterated Prisoners Dilemma

This assignment revolves around evolving strategies for the iterated prisoner's dilemma (IPD). This is a problem in which two or more players are pitted against each other in a game where they must make moves without communication, instead predicting what the the other player will do next based on their previous actions. The goal for each agent (assuming they are rational) is to obtain the greatest total reward for itself. What makes the IPD interesting is the reward matrix, which generally offers the lowest reward when both agents defect, and the highest reward for an agent when it defects and its opponent cooperates.

The problem is often framed in the context of two criminals being asked to confess to a crime. If one testifies and the other stays silent, then they will be set free, if they both testify, then neither are, but if they both stay silent, neither will go to prison [1].

There has been much work done on finding the best strategies for this problem, which will achieve the best overall fitness when pitted against many other players with different strategies. There are many strategies which have been explored and defined in terms of game theory [2]. This project will implement simple strategies represented by bit strings which react to their opponent's previous move on each round.

2 Implementation

The code for this assignment can be viewed and downloaded from the GitHub repo.

2.1 Genetic Algorithm

For this assignment, the genetic algorithm program implemented for the preceding project was not used. The reasons for this included the fact that that program was written without much generalisation in mind and so would have had to have been modified extensively anyway, as well as nearly all of the implemented operators being irrelevant for this assignment and finally all of the problems that writing the project in Python caused, such as it being excruciatingly slow to run experiments.

Being written in C, we can expect to have a much faster GA. While we would expect this project to be far less computationally intensive, it does not hurt to have faster code.

2.2 Strategy Representation

We can represent an IPD strategy with a simple bit strings. C stores these as integers. For the sake of correctness, the integers were specified to be unsigned, despite us only using a small number of bits in a 4-byte data type. The bits in the strings represent instructions for the strategy to follow. By doing this, we do can compare strategies using arbitrary length games without the need to store the entire pattern. The meaning of the representations is discussed in their relevant sections below.

2.3 Measuring Fitness

Fitness is measured by comparing a given strategy to a set of fixed strategies. For each fixed strategy, we compare the performance to the provided strategy by running a game of n rounds, while summing the reward for each player as we go. We then return the difference between the provided strategy's reward and the fixed strategy's reward in order to find its comparative performance.

This approach results in the fitness scores shown in Figure 1 for each of the eight possible strategies in a 3-bit representation. C will print the integer values of the bit strings, meaning 0 is 000 and 7 is 111. The fitness function provides a satisfiable fitness score for each of the strategies. Always defect has the highest fitness score, as it is the only strategy that has no element of cooperation, making it certain to come out on top in a simple system like this where strategic cooperation is not implemented.

```
PS C:\Users\Liam\Documents\College\4th Year\CT421 - AI\assignments\assignment2> .\ga.exe
0: 1000
1: 260
2: 20
3: -960
4: 960
5: -260
6: -20
7: -1000
best: 0
PS C:\Users\Liam\Documents\College\4th Year\CT421 - AI\assignments\assignment2>
```

Figure 1: Fitness for each of the 3-bit strategies

3 Evolution Against Fixed Strategies

3.1 Experiment

For this initial experiment, our goal is to evolve a good strategy in the search space. As previously mentioned, the fitness function will test each strategy in the population against each strategy in the fixed set. Because this experiment is conducted in a 3-bit system, we can begin by evolving a strategy against every other possible strategy (all 8), to confirm that the genetic algorithm is capable of converging on what the fitness function defines to be optimal. In this case, we would expect this to be always defect.

The bit strings in this system translate to the following encoding (reading the string from left to right):

- first bit: the first move
- second bit: the move to make if the opponent cooperated on the last round
- third bit: the move to make if the opponent defected on the last round

A value of 1 at a bit position indicates cooperation, a value of 0 indicates defection. As such, always defect is 000, always cooperate is 111, tit-for-tat is 110, etc. The fixed strategies were defined as shown in Figure 2.

```
//fixed strategies
unsigned int strats[] = {
    0b000,  // always defect
    0b001,  // 1    DDC
    0b010,  // evil tit-for-tat
    0b011,  // 3    DCC
    0b100,  // 4    CDD
    0b101,  // 5    CDC
    0b110,  // tit-for-tat
    0b111   // always cooperate
};
```

Figure 2: Fixed Strategies

As suggested in the brief, the GA was run with a population size of 50 and tournament selection with a tournament size of 5 in order to select 10 parents from each generation. The game length

for fitness was 50 rounds. In a 3-bit system, there are not a lot of crossover operations we can do. Instead, this implementation takes an elitist approach of simply duplicating the tournament winners into the next generation. The GA has mutation operator that will mutate a single, random bit of a strategy. The mutation operator is applied with a rate of 0.005 to the population.

The only remaining issue is how to approach initialising the population. For this, two approaches were taken. The first was a random initialisation, which will simply randomly fill the population each time with 3-bit strategies. The other approach was to create a population which is entirely always cooperate, the worst, most exploitable strategy. This approach is included to demonstrate how we can move from the minimum of the search space to the maximum.

3.2 Results

When we run the GA for 50 iterations, we can see that with either initialisation approach, the population converges very quickly on always defect as the optimal solution. Figure 4 shows fitness over iterations for random initialisation, while Figure 6 shows fitness over iterations when we begin with a population consisting entirely of always cooperate.

```
PS C:\Users\Liam\Documents\College\4th Year\CT421 - AI\assignments\assignment2> .\ga.exe
most common strat at start: 2
STARTING GA
DONE
ran for: 0.015s
last average fitness: 999
best strat in population: 0
most common strat: 0
PS C:\Users\Liam\Documents\College\4th Year\CT421 - AI\assignments\assignment2>
```

Figure 3: Results of random initialisation

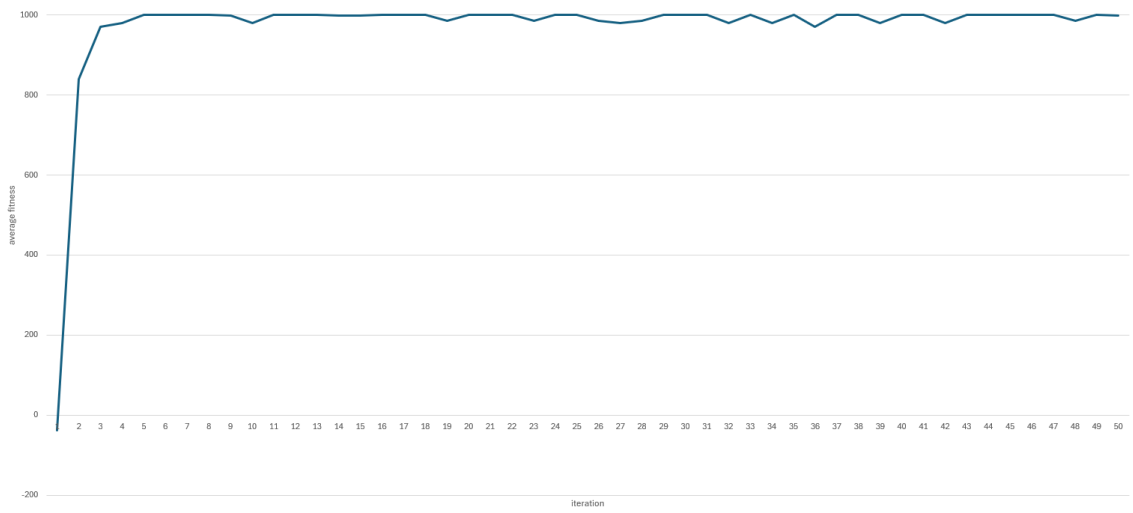

Figure 4: Fitness over iterations for random initialisation

```
PS C:\Users\Liam\Documents\College\4th Year\CT421 - AI\assignments\assignment2> .\ga.exe
most common strat at start: 7
STARTING GA
DONE
ran for: 0.014s
last average fitness: 1000
best strat in population: 0
most common strat: 0
PS C:\Users\Liam\Documents\College\4th Year\CT421 - AI\assignments\assignment2>
```

Figure 5: Results of always cooperate initialisation

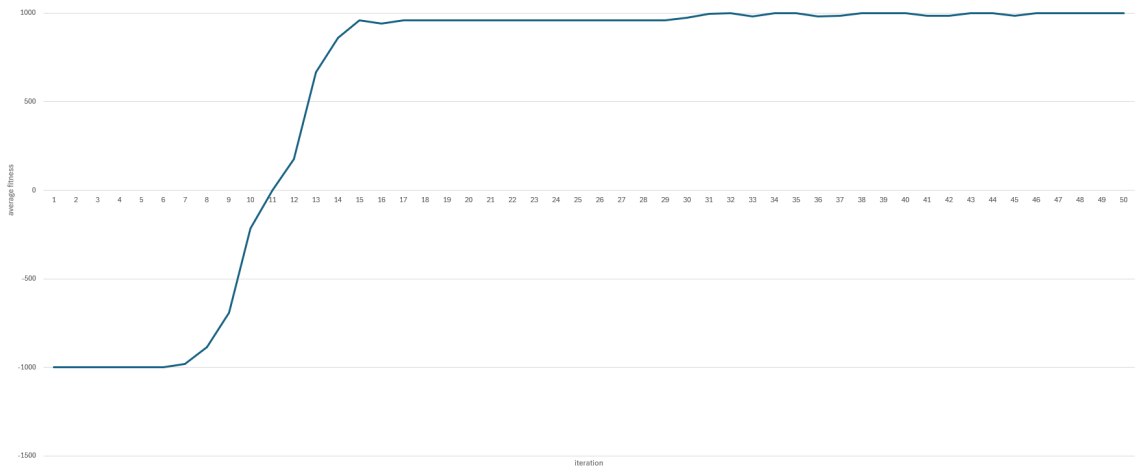

Figure 6: Fitness over iterations for always cooperate initialisation

Next, the fixed strategies were replaced with a set of all always defect. This yields the fitness for each of the three bit strategies shown in Figure 7.

```
PS C:\Users\Liam\Documents\College\4th Year\CT421 - AI\assignments\assignment2> .\ga.exe
0: 0
1: -1960
2: 0
3: -1960
4: -40
5: -2000
6: -40
7: -2000
best: 0
PS C:\Users\Liam\Documents\College\4th Year\CT421 - AI\assignments\assignment2>
```

Figure 7: Fitness for each 3-bit strategy against exclusively always defect

Interestingly, this indicates how both always defect and evil tit-for-tat have an equal chance of being evolved, since the fitness function considers them to have the same level of fitness. The results of running the GA in Figure 8 show how evil tit-for-tat was the evolved strategy. It makes sense that the program would evaluate them to have the same fitness in this case, as there is no opponent in the fixed strategy that will cooperate, essentially converting the problem to a 2-bit representation. This is why there are two of each fitness values in the test output. In this particular 2-bit system, evil-tit-for-tat becomes always defect.

```
PS C:\Users\Liam\Documents\College\4th Year\CT421 - AI\assignments\assignment2> .\ga.exe
most common strat at start: 5
STARTING GA
DONE
ran for: 0.015s
last average fitness: 0
best strat in population: 2
most common strat: 2
```

Figure 8: Results for running the GA with a fixed strategy set of exclusively always defect

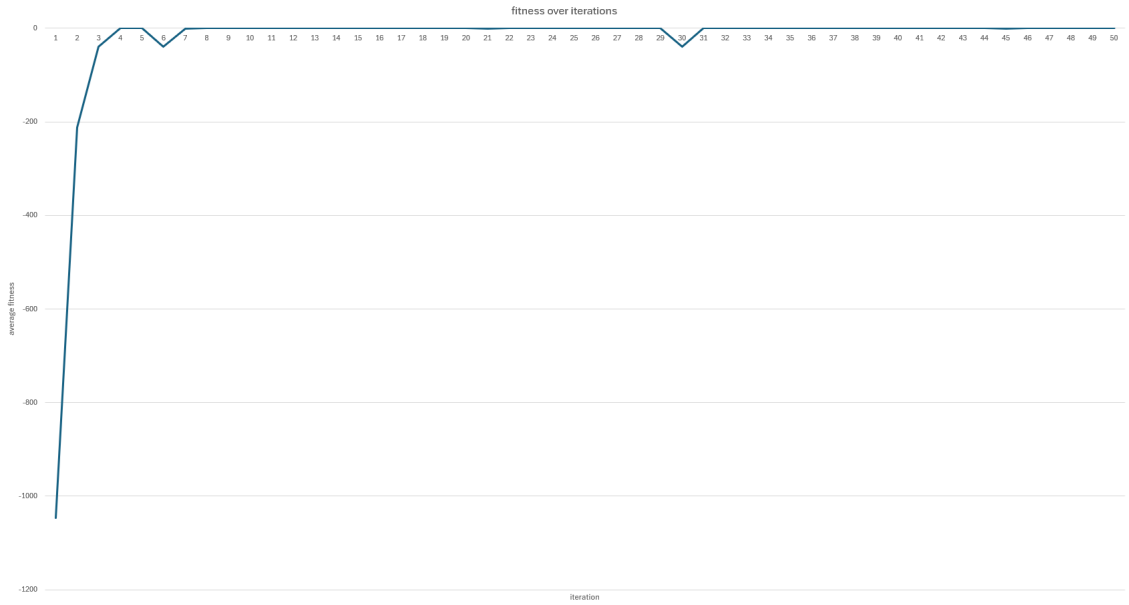

Figure 9: Fitness over iterations when running against exclusively always defect

The opposite experiment was then conducted: a fixed strategy set of only always cooperate strategies. The fitness function will return the score depicted in Figure 10 for each strategy.

```
PS C:\Users\Liam\Documents\College\4th Year\CT421 - AI\assignments\assignment2> .\ga.exe
0: 2000
1: 2000
2: 40
3: 40
4: 1960
5: 1960
6: 0
7: 0
best: 0
PS C:\Users\Liam\Documents\College\4th Year\CT421 - AI\assignments\assignment2>
```

Figure 10: Fitness for each 3-bit strategy against exclusively always cooperate

Once again, the representation is essentially reduced to a 2-bit system in the eyes of the fitness function. Running the GA produces the results in Figure 11.

```
PS C:\Users\Liam\Documents\College\4th Year\CT421 - AI\assignments\assignment2> .\ga.exe
most common strat at start: 6
STARTING GA
DONE
ran for: 0.013s
last average fitness: 2000
best strat in population: 1
most common strat: 1
PS C:\Users\Liam\Documents\College\4th Year\CT421 - AI\assignments\assignment2>
```

Figure 11: Results from running the GA with exclusively always cooperate in the fitness strategies

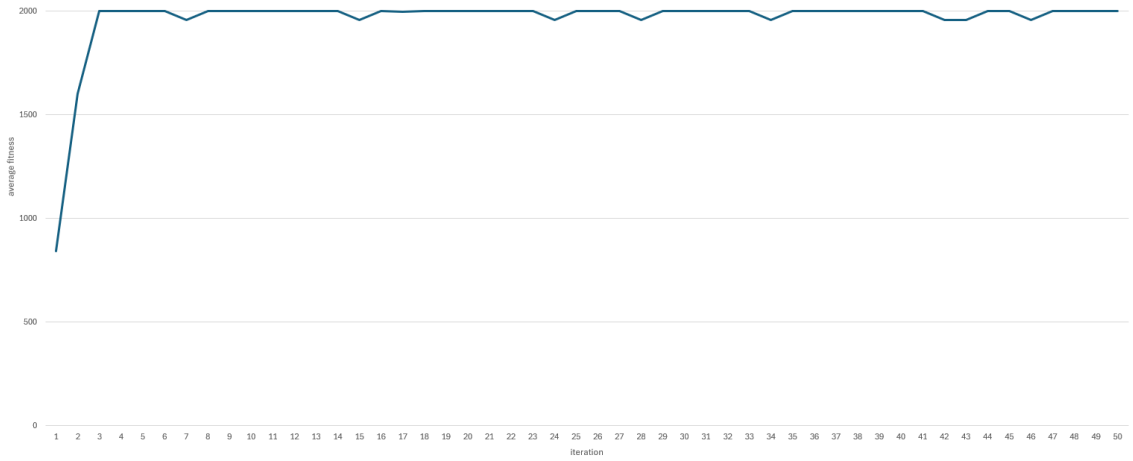

Figure 12: Fitness over iterations when running against exclusively always cooperate

Overall, changing the fixed strategy set had essentially no impact on the GA's preference for always defect as the best strategy.

4 Extension

4.1 Experiment

For the next part of the project, we will extend the memory of the strategies to two rounds, to allow them to make more complex decisions. This is achieved by adding another two bits to the strategies, which will enable them to take specific actions if their opponent repeats a move for two turns. The new bits indicate either a defection or a cooperation for two bits of cooperation or defection.

This extension should allow for more complex strategies and potentially longer-term planning in the games. However, it has been shown that more complex strategies do not have any advantage over simpler ones [3].

This increases the representation to be a 5-bit model. As such, there are now 32 different possible strategies in the system. Given that C stores these bit strings as integers, we can still easily check the corresponding fitness for each, as shown in Figure 13. The fitness function itself was also updated to measure performance of strategies against all strategies. I.e., all strategies are still included in the fixed set. This system is also compatible with the previous 3-bit system, meaning the representations of the first eight strategies are the same as those used in the previous experiment.

In this system, the first two bits indicate whether the strategy should cooperate or defect when their opponent has either cooperated or defected for two rounds in a row. The remaining three bits are the same as before: the third bit is the first move, the fourth what to do in response to a cooperation, the fifth what to do in response to a defection.

```
PS C:\Users\Liam\Documents\College\4th Year\CT421 - AI\assignments\assignment2> .\ga.exe
0: 5060
1: 1395
2: 85
3: -2960
4: 4880
5: 255
6: -75
7: -3100
8: 4060
9: 1395
10: 1140
11: -635
12: 3335
13: 255
14: 305
15: -1275
16: 985
17: 105
18: 85
19: -635
20: 855
21: -70
22: -75
23: -740
24: -210
25: -2245
26: -1505
27: -2960
28: -540
29: -2400
30: -1670
31: -3100
PS C:\Users\Liam\Documents\College\4th Year\CT421 - AI\assignments\assignment2>
```

Figure 13: Fitness scores for each strategy in the 5-bit system

This shows how even still, always defect comes out on top as the most effective strategy in the fitness function. When we run the GA, we see in Figure 14 that it still converges quickly on always defect. This result supports previous research that more complex solutions do not do any better than simple ones, although a 5-bit representation is not too complex either [3]. The results of the previous experiment have simply been somewhat expanded; 00100 is still in second, for example.

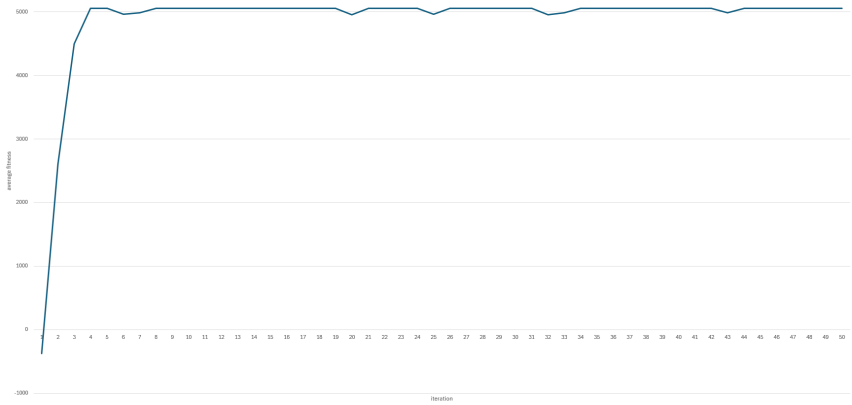

Figure 14: Fitness over iterations for the extended representation

5 Conclusion

In this project, the iterated prisoners dilemma was implemented into a GA to see if we could evolve the optimal strategy for agents to obtain the highest reward. The GA was successfully able to do this for both a 3-bit and 5-bit system based on the fitness function definition. Always defect was identified by the fitness function as being the strongest strategy in both systems when pitted against every other possible strategy.

Based on previous research, we might have expected tit-for-tat to come out on top; its ability to draw ultimately winning in the end [3]. However, previous research has produced similar results when testing against all possible strategies [4]. If we were to improve this approach, the way to go about that would be to implement more complex strategies, such that they can have behaviours

similar to that of GRIMM, or others which have specific instructions [2]. We are missing these kinds of strategies from this implementation, which are capable of further cooperation without being taken advantage of to a significant degree.

In this system, it is clear that defection is king. The fitness function results and the GA results show how the more cooperation in the strategy, the easier it is for you to be exploited.

References

[1] William Poundstone. *Prisoner's Dilemma*. New York: Anchor, 1993.

[2] Steven Kuhn. *Strategies for the Iterated Prisoner's Dilemma*. 2019. URL: https://plato.stanford.edu/entries/prisoner-dilemma/strategy-table.html.

[3] *The Axelrod Tournaments*. 2011. URL: https://lawrules.wordpress.com/2011/09/05/the-axelrod-tournaments/.

[4] Stuart Ferguson. *Prisoner's Dilemma Revisited*. 2024. URL: https://medium.com/@metaform3d/prisoners-dilemma-revisited-bfdaa0e02c80.