Iterative Prisoner's Dilemma

An in-depth analysis of ten human-designed strategies and three optimizations for COMP 3710

1st Aran Abrahamlingam
School of Computer Science
University of Windsor
Windsor, Canada
abrah11l@uwindsor.ca

2nd David Dagati
School of Computer Science
University of Windsor
Windsor, Canada
dagati@uwindsor.ca

3rd Gabriel Rueda
School of Computer Science
University of Windsor
Windsor, Canada
ruedag@uwindsor.ca

***Abstract*—This experiment looks in depth at the Iterated Prisoner's Dilemma, considering ten human-designed strategies as well as three optimization algorithms to find the best strategy for a player in the IPD. The goal is to compete one strategy against the other and have one strategy win against the other by having a lower score, representing the lower year count of jail time. From the genetic algorithm optimization, we reviewed the concept of population and memory where multiple tests were run modifying these variables to reach the best outcome. We learned that a moderate population of 128 and a high memory of 3 previous turns produces the best outcome. For the hill climbing algorithm tests were run on different arrays representing player decisions. The most interesting result was the high number of wins cooperating every round except the last had. From exhaustive, we ran all ten strategies against each of the ten strategies resulting in 100 rounds, where each strategy was used 20 times. These tests were run five times with an increasing number of rounds each time. This resulted in a consistent top three performers in All Defects, Adaptive Pavlov, and Suspicious Tit For Tat. These three optimizations show us how many approaches there are to participate in the IPD and how each approach will result in a player receiving a different outcome.**

***Index Terms*—strategies, optimization, genetic, hill climbing, exhaustive**

I. Introduction

In this project, we will explore the iterated prisoner's dilemma. This is a classic game theory problem where two players repeatedly play a game against each other. The goal of each player is to maximize their own reward, which is based on their own and their opponent's actions. The players must decide whether to cooperate, which benefits both players, or to defect, which benefits one player at the expense of the other. If they both cooperate then they both get 1 year of prison time, if they both defect then they both get 10 years but if one cooperates and the other defects then they get 20 and 0 years.

In this project, we aim to explore and compare the performance of different optimization methods and strategies. To achieve this goal, we first created a main simulator that can have different strategies compete with each other. We coded 10 human-designed strategies, including Tit for Tat, Pavlov, Adaptive Pavlov, Suspicious Tit for Tat, GRIM, Tit for Two Tats, Random, All cooperate, All defect, and Every other. We used different optimization methods, including Genetic Algorithm, Hill Climbing, and Exhaustive Search, to find the optimal parameters for each strategy. These methods explore the solution space by iteratively evaluating and improving candidate solutions. We also investigated how different factors such as population size, mutation probability, and memory depth affect the performance of each strategy.

The optimization methods we used in this project are commonly used in AI and machine learning applications, and they have been shown to be effective in a wide range of scenarios. Genetic Algorithm, for example, is a population-based method that imitates the process of natural selection to search for good solutions. Hill Climbing is a gradient-based method that iteratively improves the current solution by making small changes to its parameters. Genetic Algorithm and Hill Climbing are heuristic-based methods that search for good solutions but do not guarantee optimality. Exhaustive Search, on the other hand, evaluates all possible solutions to find the optimal one. By comparing these methods, we can gain insights into which methods are most suitable for different scenarios.

In this report, we will go in-depth on the methodology, experimental setup and the results we obtained as well as a literature review on a 2004 Prisoner's Dilemma competition. We will discuss how each optimization method performed and how it affected the performance of the strategies. We will also compare the different human-designed strategies and discuss which ones performed the best. By comparing the different methods and strategies, we can gain insights into how to achieve better outcomes in competitive scenarios.

II. Literature Review

In 1979, Axelrod organized a competition using the prisoner's dilemma. He gathered 64 different strategies, including one random, and had them compete against each other. While in a single round, defection seems to be the optimal solution, Axelrod's experiments used multiple iterations. This resulted in cooperative strategies receiving better results.

The winner of this competition was Tit-For-Tat (Described in Section III. Strategies). Axelrod came up with a few criteria for a successful strategy. The strategy should not attempt to gain more points using defection. The strategy should not be the first to defect The strategy must quickly respond to consequences and defections. It must be clear on its consequences

and adapt to the opponent's strategy. The strategy should not try to "outsmart" its opponent. A complicated strategy could lead to the strategy thinking the opponent is RANDOM.

One of the reasons these cooperative strategies succeeded is that "Kingmaker" strategies participated in the competition. These strategies prioritize defection to achieve a more optimal score. The existence of these strategies allows cooperative strategies like Tit-For-Tat to succeed. One example is the GRIM strategy, which is a "kingmaker" strategy that will be analyzed below.

With the optimizations of the Genetic Algorithm and Hill Climbing, it can be shown whether the criteria proposed will make those strategies defined optimal.

In 2004 and 2005 the competition was executed again, but with more strategies. One of the strategies was presented by Southampton's School of Electronics and Computer Science (ECS). The team proposed a solution in which multiple of its players will communicate during the competition. The strategy will start with a sequence of moves to determine its identity. Then if both players identify each other, they will transmit messages to improve their performance.

With the optimization of Genetic Algorithms, these principles can be applied. In the genetic algorithm, multiple strategies are played. The offspring will be developed to get an optimal solution. The idea is similar to how ECS used communication. Both optimizations will use evolutionary techniques to learn the strategy of the opponent.

III. Strategies

The following ten strategies were used in this study. Each strategy brings a unique approach to selecting an option of either Cooperating or Defecting (Depicted as 0 and 1 respectively) in the Prisoner's Dilemma experiment.

All Cooperate: Will Always Cooperate.

All Defect: Will Always Defect.

Every Other: Will switch back and forth between Cooperating and Defecting, beginning with cooperation.

GRIM: Will cooperate until the opponent defects once, in that case, it will then always defect.

Pavlov: Will cooperate if the opponent selected the same option last round, beginning with cooperation.

Random: Will select to either cooperate or defect randomly.

Suspicious Tit for Tat: Will start with defecting, but then copies the opponent's previous decision on every other round.

Tit for Tat: Will start with cooperation, but then copies the opponent's previous decision on every other round.

Tit for Two Tats: Will Cooperate unless the opponent defects twice in a row.

Adaptive Pavlov: Will begin with Tit for Tat for the first six rounds. Then will place the opponent into one of five categories based on their response, which will then select an optimal strategy based on the category they are placed in.

IV. Genetic Algorithm

A. General Overview

A genetic algorithm consists of the following:

Fitness: Each solution in the population is evaluated by competing against 8 of the 10 strategies (all except adaptive Pavlov and GRIM). Adaptive Pavlov and GRIM will be used to test our genetic algorithm results. After competing against the 8 strategies, the sum of the scores will be their fitness value. The number is divided by 10 to get smaller values.

Selection: After each solution's fitness is calculated, the population is ordered from lowest to highest fitness. From there, the first 32 solutions will move on to the crossover phase. These will be known as the "best solutions."

Crossover: Each element has a crossover probability of 0.7. Basically, each element has a 70% chance that it will keep its original value. Otherwise, it will get that value from one of the best solutions, which is chosen randomly.

Mutation: The following below do not have any mutations. The idea of mutations will be explored later in this section.

The genetic algorithm consists of three different types: no memory, memory of the past move and memory of 3 past moves. Within these three memory types, we will see what population size is optimal.

B. Finding an Optimal Memory Depth

1) No Memory Depth: Figure 1 shows the average score that each population achieves during each generation from 0 to 100. The average score is calculated by the average of all the fitness values from the entire population.

All five test cases in the graph are all with no memory depth. A population of 32 or 128 seems to be effective enough. A population of 32 will be chosen for reliable results.

Also, at around 15-20 generations, the score plateaus. Thus, a generation count of around 40-50 should be sufficient for this memory and population size.

2) One Memory Depth: Each solution is an array of 10 elements, where the "i"th element represents the "i"th move. First move is the action the algorithm will take. The second move will be an array of format [a, b], where a is the action that the algorithm will take if the opponent previously played "0" and b is the action that the algorithm will take if the opponent previously played "1".

Figure 2 demonstrates the average scores achieved by the entire population at each generation count for a memory depth of one. A memory depth of one experienced an optimal population of above 16. Again, a population of 32 will be used to provide reliable results.

At around 15-20 generations, the score levels off. Thus, a generation count of 40-50 should be sufficient for a memory depth of 1.

3) Three Memory Depth: Each solution is an array of 10 elements, where the "i"th element represents the "i"th move. First move is a single value and second move are two values (like One Memory Depth). The third move has the following format: [00, 01, 01, 11]. For each element, the first digit is the opponent's move 1 play ago and the second digit is the move 2 plays ago.

The remaining elements have the following format: [000, 001, 010, 011, 100, 101, 110, 111]. For each element, the first

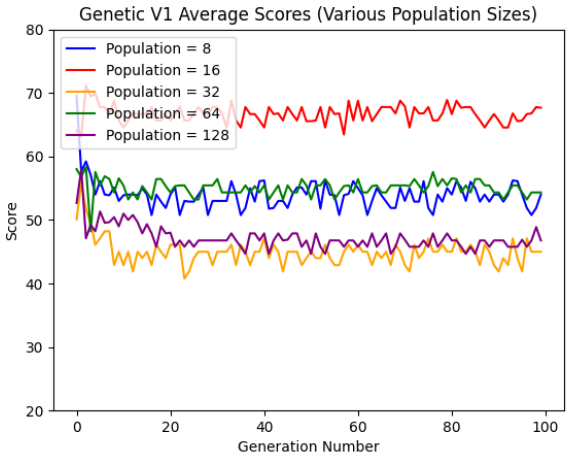

Fig. 1. Testing Population Sizes with No Memory Depth.

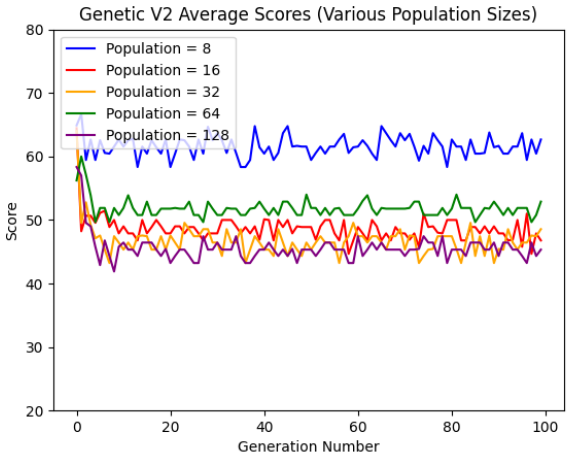

Fig. 2. Testing Population Sizes with Memory Depth of One.

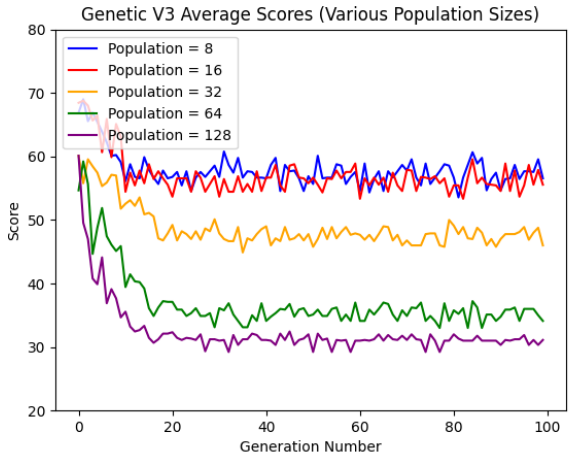

Fig. 3. Testing Population Sizes with Memory Depth of Three.

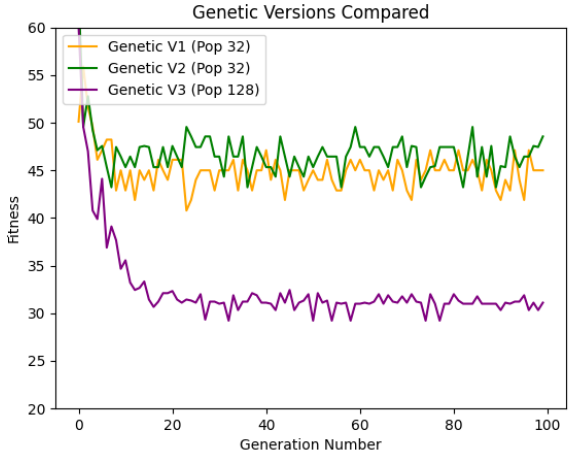

Fig. 4. Side by side comparison of all the memory depths.

digit is the opponent's move 1 play ago, the second digit is the move 2 plays ago and the third digit is the move 3 plays ago. The value in this 8-element array is the action that algorithm will take if the order of those moves were taken.

According to Figure 3, for a memory depth of three, the population size needs to be a bit higher. The optimal population count is 128, since it will result in the lowest scores.

Additionally, at around 30-40 generations, the score plateaus. Thus, a generation count around 100 would provide enough time for the algorithm to stabilize.

4) Choosing a Memory Depth: After analyzing the memory depths of none, one and three, the genetic algorithm with a memory depth of three is superior. We tested a population of 32 for both Genetic V1 and Genetic V2 and a population of 128 for Genetic V3.

Figure 4 demonstrates the importance of memory for one of these strategies. For example, the Tit For Tat strategy is a well-performing strategy, but is limited to a memory depth of one. This graph proves that a memory depth of three has the potential to create more optimal strategies.

C. Modifying the Genetic V3 Algorithm

1) Implementing Mutation: A mutation may or may not be needed for this Genetic Algorithm. No mutation has been previously used but now mutation probabilities of 0.01, 0.02, 0.05, and 0.1 will be tested. However, the results achieved are very volatile and the results are unreliable, as shown in Figure 5.

2) No Random Play: To resolve the volatility in the results for adding mutation, the random strategy was removed from the Algorithm. While the random strategy may prepare the genetic algorithm for any random value, it seems to do more harm to the algorithm. Firstly, the random strategy may mislead the algorithm towards a less optimal solution against the other strategies. Secondly, after implementing a mutation, a random strategy will not be needed to increase the range

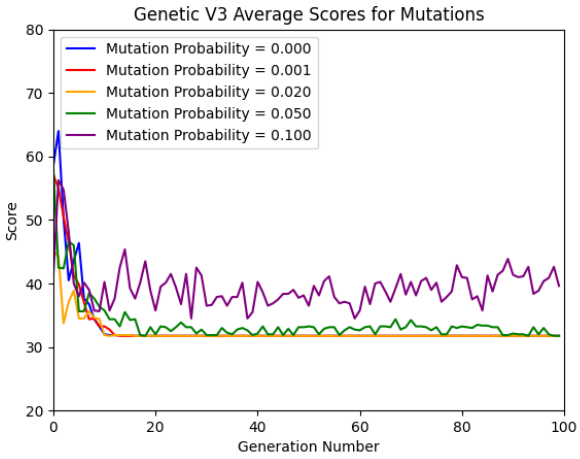

Fig. 5. Different mutation probabilities tested.

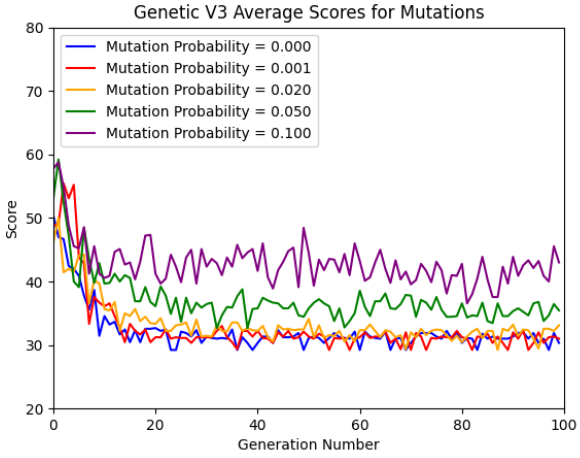

Fig. 6. Different mutation probabilities tested but with no random strategy.

of possible solutions. A mutation will allow for the value to randomly change to a solution never found before.

The results in Figure 6 demonstrate the benefit of removing the random strategy in favour of a mutation probability. However, this is not the case for all mutation probabilities. The mutation probability of 0.100 and higher fails to achieve optimal results. Also, the higher the mutation probability, the more volatility the score result will have. For example, a mutation probability of 0.100 (purple line) is volatile to the point in which it is plateaus higher than the other mutation probabilities.

The optimal mutation probability could be anything from 0.000 to 0.020, since the probabilities in that range level off at around the same movement. Thus, 0.001 will be used for our final result.

3) Examine Crossover Probability: For this test, we will use Genetic V3 with a population of 128 and a mutation probability of 0.001, since it demonstrates the most optimal

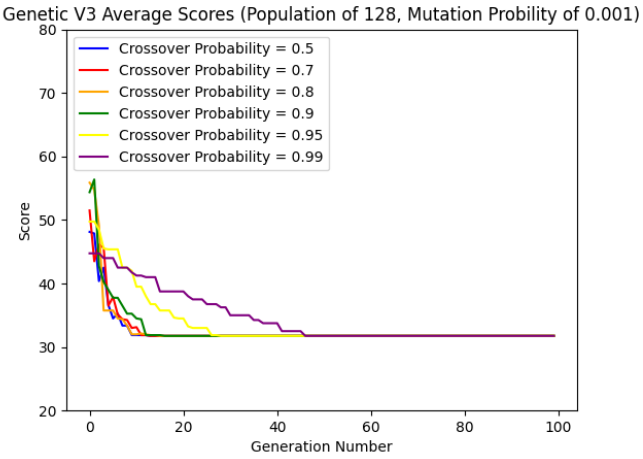

Fig. 7. Different crossover probabilities tested.

results. Crossover probability is the probability that each value in the array will keep its original value. Otherwise it will get its value from another solution. There will also be no random strategy in these tests.

According to Figure 7, The crossover probability will only determine the slope a strategy takes to obtain a result. All the crossover probabilities will end up levelling off at the same score and obtaining the same result array. However, a crossover probability of 0.5 will be chosen since it obtains the final result the quickest.

D. Final Genetic Algorithm

1) Specifications:

- Resulting Array: [0, [0, 0], [0, 0, 1, 1], [0, 0, 1, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 1, 0, 1], [0, 0, 1, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 1, 0, 1], [0, 0, 1, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 0, 1], [1, 0, 1, 0, 0, 0, 0, 1]]
- Memory Depth: 3
- Population: 128
- Crossover Probability: 0.5
- Generation Count: 100
- Mutation Probability: 0.001
- Random Strategy: No

According to Table 1, the score difference is positive (or 0) if the Final Genetic V3 succeeded and negative if Final Genetic V3 failed. Seven strategies succeeded, and three strategies failed. One of the strategies that failed is all defects, which isn't a practical strategy and would not be used in a real life scenario.

However, the final genetic strategy failed against Adaptive Pavlov, which wasn't in the training set of strategies. However, the Adaptive Pavlov is a strategy that uses a memory depth of six, which will place the final genetic strategy with a memory depth of three at a disadvantage. The final genetic strategy proposed could have probably won against the Adaptive Pavlov strategy if it used a larger memory depth and included Adaptive Pavlov in the training set.

Strategy Name	Genetic V3 Score	Opponent Score	Score Difference
Tit for Tat	9	29	20
Pavlov	9	29	20
Adaptive Pavlov	76	16	-60
Suspicious Tit for Tat	28	28	0
Tit for Two Tats	9	29	20
All Cooperate	9	29	20
All Defect	120	80	-40
Every Other	61	121	60
GRIM	9	29	20

TABLE I
RESULTS OF FINAL GENETIC STRATEGY AGAINST THE 9 STRATEGIES

Finally, the genetic strategy proposed did well overall and could win against most strategies. However, including more strategies in the training set and larger memory depth could have given the genetic strategy a more diverse set of strategies that it could defeat.

V. HILL CLIMBING

A. General Overview

Hill Climbing is an optimization algorithm that can be used to find optimal solutions to a given problem. It is a local search algorithm that starts with an initial solution and iteratively moves to a neighbouring solution that has a better score. This process continues until a local maximum is reached, which is the best solution that can be obtained using the given algorithm. However, it can also get stuck in local maxima and fail to find the global optimum. To overcome this issue, we use a technique called random restarts to explore more of the solution space and avoid getting stuck in local optima.

To apply this technique to the iterated prisoner's dilemma, we can use the main simulator we created to have the different strategies compete with each other. For each iteration, we would input the array of random numbers with sizes 5 and 10 as the parameters for the strategy and then run the simulation to obtain the score of the strategy against all the human-designed strategies. We would then modify one value in the array and rerun the simulation to see if the new array results in a better score. We then tested to see if iterating over the same array (5, 10, 50 times) would improve the array and converge to the optimal array. Once we have found the array that results in the best score, we can use the random reset technique to explore the solution space further. This involves randomly resetting the parameter values to a new position/ random array and repeating the hill-climbing process to see if we can find a new array that performs better. We utilize bar graphs to represent the percentage of times each array wins, and we compare the different strategies to determine the optimal solution. The two different techniques used are the array with the lowest score and the most wins.

B. Optimal strategy for Lowest Score

On Figures 8 and 9, for an array of size 5, array: [00001] only lost 4% of the time. For an array of size 10, array: [0000000001] is the best and array:[0000000111] and [0000000011] in 2nd and 3rd. While this doesn't initially give us the optimal solution, we can see the pattern is to always cooperate and defect on the last move. This pattern emerges

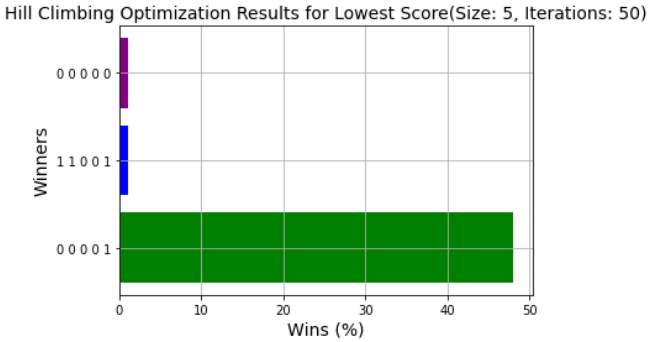

Fig. 8. Optimal Strategy for Lowest Score. Size 5, Iterations 50

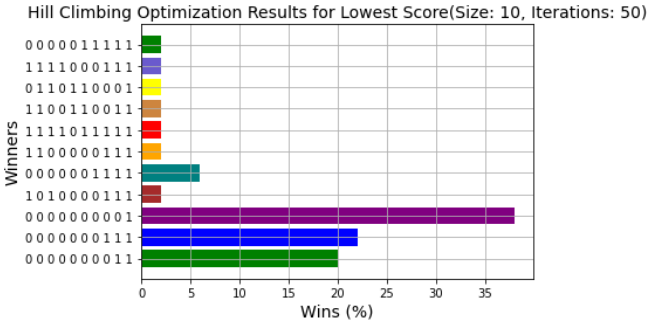

Fig. 9. Optimal Strategy for Lowest Score. Size 10, Iterations 50

because most human-designed strategies are cooperative and retaliate when defected. The strategy of defecting at the end allows the opponent no opportunity to retaliate and thus is the best way to minimize points. This strategy would be the best to accumulate the least amount of points.

C. Optimal strategy for Highest Score

On Figures 10 and 11, we take a different approach and find the array that leads us to win the most games. By a majority in both graphs, the best strategy is to always defect. The issue with this optimization method is that it runs into many local maxima and gets stuck. A trend is observed that it usually

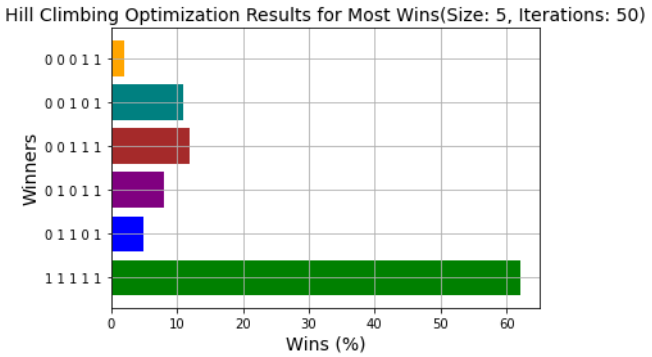

Fig. 10. Optimal Strategy for Most Wins. Size 5, Iterations 50

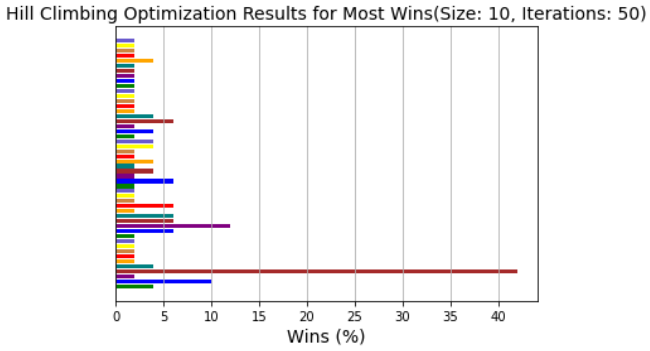

Fig. 11. Optimal Strategy for Most Wins. Size 10, Iterations 50

ends with a score of 9/10 or 10/10, 10/10 being the strategy to always defect.

D. Conclusion

Overall, the two optimization methods allow us to explore the optimal strategy for different goals in the iterated prisoner's dilemma. Discovering how drastically different the optimal strategy is for the problem on hand. If the problem is to get the least points then the optimal strategy is to always cooperate and defect on the last move. If the problem is to win the most games then the optimal strategy is to always defect so that you could never lose. In the appendix, we show more results of this method showing different iterations.

VI. Exhaustive Search

A. General Overview

The purpose of the Exhaustive algorithm is to look into every possible combination of our ten selected strategies. This results in 100 combinations where each algorithm will compete for the lowest score against any of the ten algorithms. In this method there is no optimization for finding the best results in a short amount of time, rather the focus is on the number of times each algorithm is able to best another.

In this experiment, a trial would begin by receiving a name and a number of rounds. Next, the matchups were created by nesting for loops which assigned Player A and Player B with one of the ten strategies. Then the simulator object we designed was used to simulate each matchup. When the number of rounds was met the simulation would finish and scores for each player would be analyzed and a winner would be selected. This lower score winner would then gain a point as shown in each Trial's Exhaustive winner count. In the case of a Tie, a point would be allocated to a Tie column.

Five trials were conducted to understand a sense of each algorithm's performance over short and long periods of time. Trial 1 took place over 10 rounds, Trial 2 used 25 rounds, Trial 3 doubled to 50 rounds, Trial 4 doubled again to 100 rounds, and lastly, Trial 5 was raised to 500 rounds.

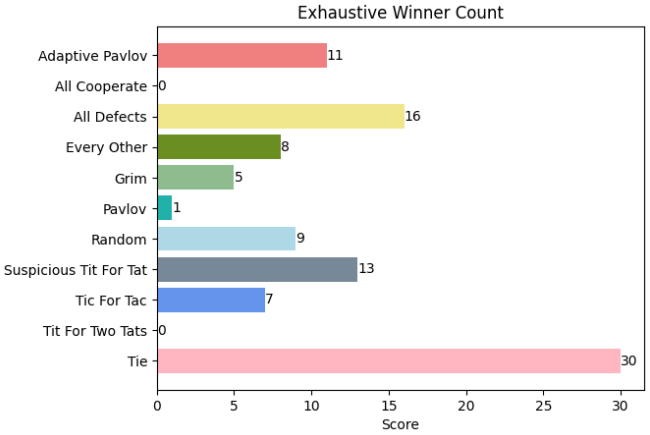

Fig. 12. Number of Rounds = 10

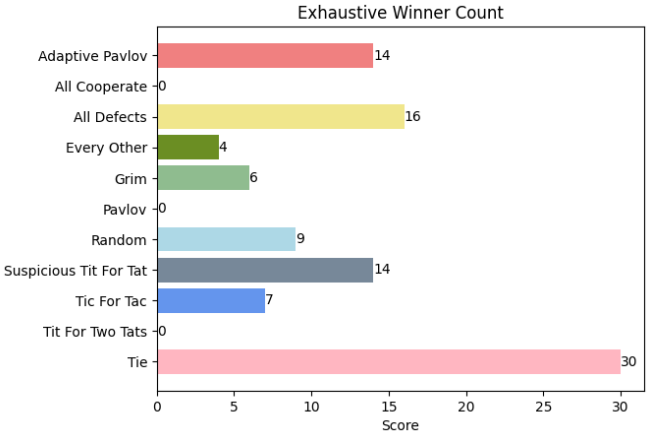

Fig. 13. Number of Rounds = 25

B. Trial 1

In Figure 12 we can see how most strategies are able to win multiple matchups. This however is not the case for All Cooperate, Tit for Tat, or Tit for Two Tats, as they were not successful in any of their 20 rounds (Ten consecutive rounds as player 1, and ten separate rounds as player B) this figure also reveals that a Tie is the most common occurrence across any winner. Ignoring the tie All Defects is the best-performing strategy in a Trial of 10 rounds.

C. Trial 2

Figure 13 takes a look at Trial 2. This second trial shows how small changes are made when the number of rounds is slightly increased, with the largest increase being a tie between Random, and Suspicious Tit for Tat by 3 points. While the largest decrease is Every Other losing 6 points.

D. Trial 3, Trial 4, and Trial 5

Trial 3 and Trial 4 increased the number of rounds to 50, and 100 respectively. Other than observing the peak of Ties overall Trials in Figure 16, the data remains very similar, and there are no major changes observed in these trials.

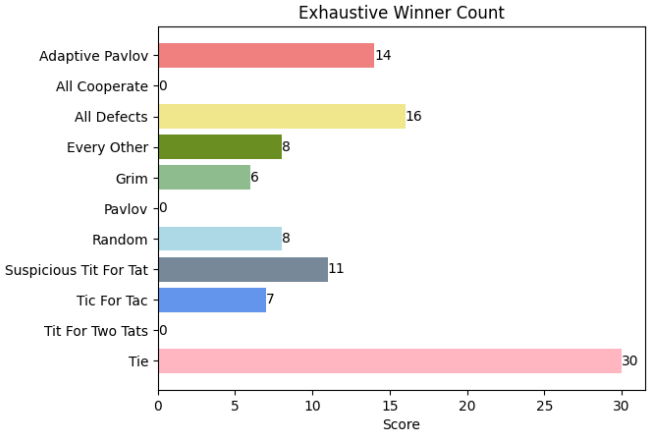

Fig. 14. Number of Rounds = 50

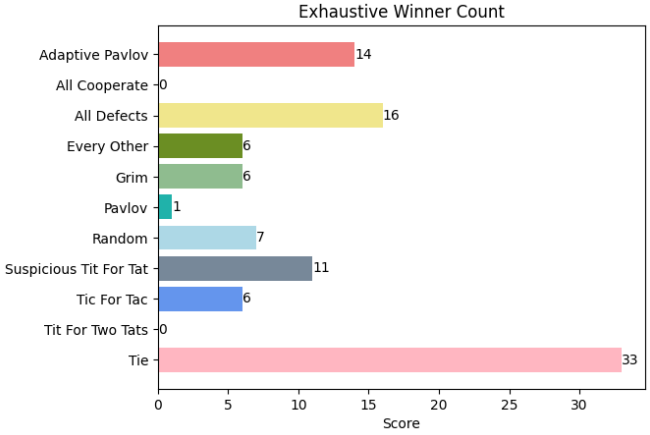

Fig. 15. Number of Rounds = 100

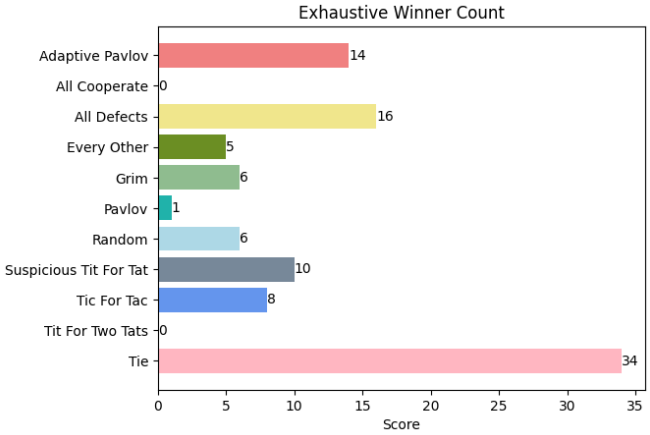

Fig. 16. Number of Rounds = 500

Finally, Trial 5 has the largest increase in the number of rounds. Comparing these results to those which come before it there are three key facts to notice:

- A tie is always the most common result when one strategy is used directly against another
- When ignoring a Tie the top 3 strategies always include:
  - All Defects
  - Adaptive Pavlov
  - Suspicious Tit For Tat
- All Cooperate, Tit for Tat, and Tit for Two Tats never win in any Trial

E. Conclusion

These three key points summarize the results of the Exhaustive algorithm's research. In the future to expand the results, more strategies could be added to increase the total count to over 100. This would give the old strategies a chance to win against new strategies or would allow for the new strategies to collect their own wins.

VII. Conclusion

In conclusion to this report, we will review the advantages of using the three optimizations of Genetic, Hill climbing, and Exhaustive search. From Genetic we learned how population and memory can affect existing strategies, improving them over their original forms. With a larger memory the algorithm is able to return stronger decisions resulting in an optimal score. From analyzing the Hill Climbing optimization algorithm we learned how always defecting will result in a high number of wins. This is supported by its results in the Exhaustive search, where the all defect strategy always comes out in the top 3 across all five tests. Exhaustive search also supports the results from genetics as adaptive pavlov is also one of the top performing strategies in each Trial.

Across the three optimizations, there were many tests run modifying key variables in the Iterated Prisoner's Dilemma. The population and memory of the genetic algorithm, the modification of the optimal array in the hill climbing algorithm, and the number of rounds in the exhaustive search are all manipulated to return their idea of the optimal strategy to maximize a player's wins in the iterated players' dilemma.

References

[1] University of Southampton Team wins prisoner's dilemma competition (October 7, 2004) University of Southampton. Available at: https://www.southampton.ac.uk/news/2004/10/team-wins-competition.page (Accessed: February 15, 2023).

[2] M. Jurišić, D. Kermek and M. Konecki, "A review of iterated prisoner's dilemma strategies," 2012 Proceedings of the 35th International Convention MIPRO, Opatija, Croatia, 2012, pp. 1093-1097.

[3] Kuhn, Steven, "Prisoner's Dilemma", The Stanford Encyclopedia of Philosophy (Winter 2019 Edition), Edward N. Zalta (ed.), URL = https://plato.stanford.edu/archives/win2019/entries/prisoner-dilemma.

VIII. Appendix

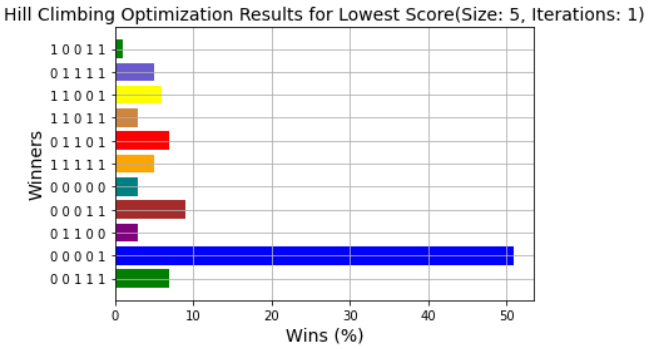

Fig. 17. Optimal Strategy for Lowest Score. Size 5, Iterations 1

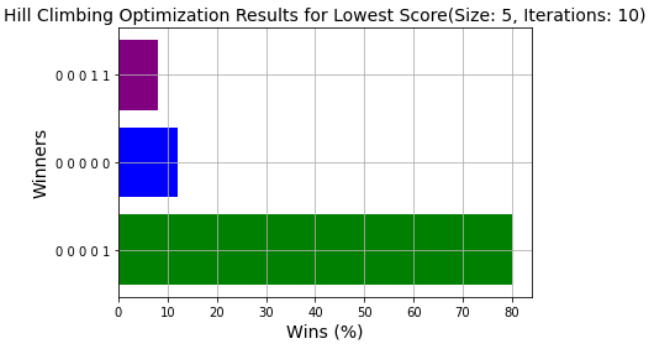

Fig. 18. Optimal Strategy for Lowest Score. Size 5, Iterations 10

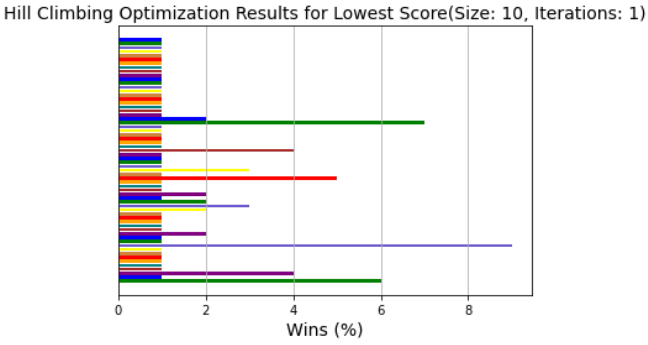

Fig. 19. Optimal Strategy for Lowest Score. Size 10, Iterations 1

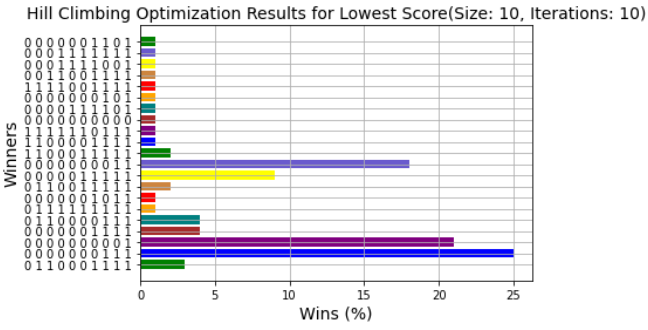

Fig. 20. Optimal Strategy for Lowest Score. Size 10, Iterations 10

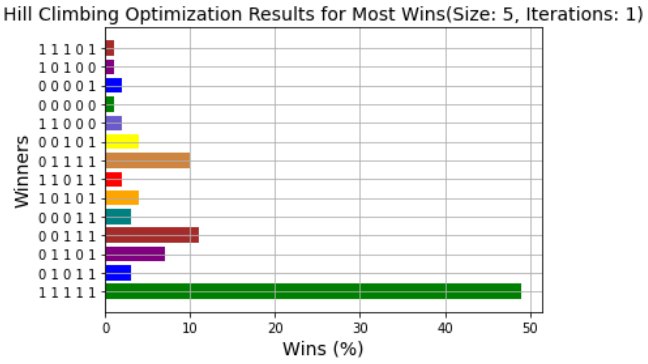

Fig. 21. Optimal Strategy for Most Wins. Size 5, Iterations 1

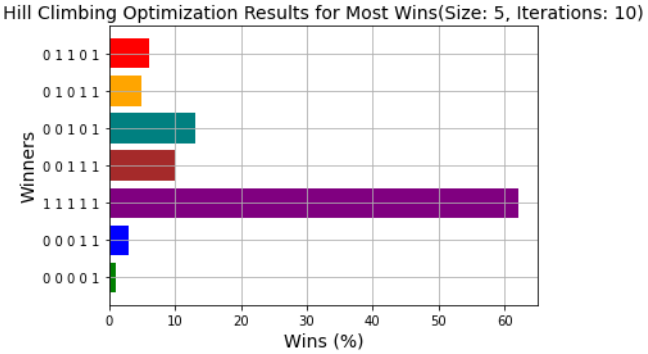

Fig. 22. Optimal Strategy for Most Wins. Size 5, Iterations 10

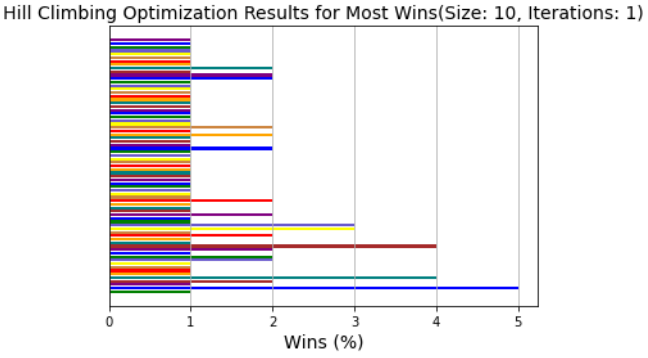

Fig. 23. Optimal Strategy for Most Wins. Size 10, Iterations 1

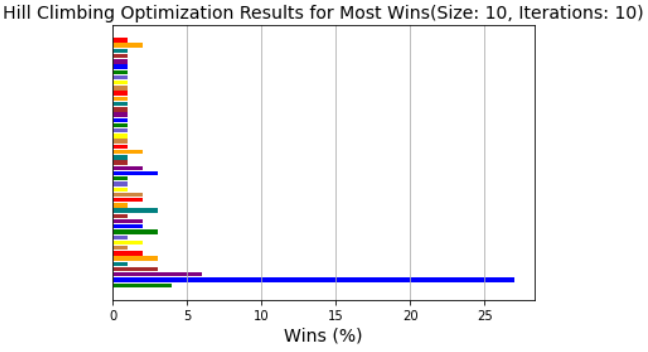

Fig. 24. Optimal Strategy for Most Wins. Size 10, Iterations 10