A Comparative Study of Optimization Algorithms and ML Techniques for Prisoner's Dilemma

Matt Davies
dept. of Computer Science
University of Windsor
Windsor, Canada
davies62@uwindsor.ca

Jolin Lodhiya
dept. of Computer Science
University of Windsor
Windsor, Canada
lodhiya@uwindsor.ca

Sara Sarhill
dept. of Computer Science
University of Windsor
Windsor, Canada
sarhills@uwindsor.ca

Zhuqing Sha
dept. of Computer Science
University of Windsor
Windsor, Canada
shaz@uwindsor.ca

Abstract—**This paper presents a comparative study of several optimization algorithms and machine learning techniques applied to the Iterated Prisoner's Dilemma (IPD). We investigate the evolution of strategic behaviour by employing local search methods, including Hill Climbing, Simulated Annealing, Tabu Search, and Genetic Algorithms, to evolve effective strategies over multiple rounds of play. The study evaluates the fitness landscape by aggregating payoffs across different configurations and monitoring the evolution of strategies through rolling averages of performance metrics. Furthermore, we incorporate supervised machine learning models to classify evolved strategies as "good" or "bad" based on their performance and cooperation rates. Our experimental results indicate that local search methods, particularly Hill Climbing and Simulated Annealing, achieve higher final fitness values in a computationally efficient manner, while Tabu Search shows consistent yet slightly lower performance. In contrast, the Genetic Algorithm demonstrates significant variability and higher computational overhead. The analysis reveals that reciprocating and conditionally cooperative strategies, such as Tit-for-Tat and Win-Stay, Lose-Switch, tend to dominate over purely cooperative or defecting approaches. Overall, our findings provide insights into the trade-offs between solution quality and computational efficiency in evolving strategies for the IPD, and they pave the way for future research integrating advanced deep reinforcement learning methods and improved evaluation metrics.**

1. Introduction

In this paper, we investigate the Prisoner's Dilemma, which is a famous problem in Non-Cooperative Game Theory. Originally formulated by Merrill Flood and Melvin Dresher in the 1950s, it describes a scenario where two criminals are arrested and interrogated separately, unable to communicate with each other. Each prisoner faces a choice: to cooperate and accuse the other or to remain silent. If both confess, they each receive a 10-year sentence (-10). If one confesses while the other remains silent, the confessor is set free while the silent prisoner receives a harsh 20-year sentence (-20). However, if both choose to remain silent, they each receive a light one-year sentence on a lesser charge (-1) [2]. The dilemma arises because, while mutual silence leads to the best collective outcome, rational self-interest encourages each prisoner to confess, fearing the worst-case scenario of receiving 20 years if they remain silent while the other confesses. This tension between individual incentives and cooperative outcomes makes the problem a cornerstone of Non-Cooperative Game Theory [1].

The following payoff matrix can be respresent the prisoner's dilemma:

		John	
		cooperate	defect
Emma	cooperate	-1, -1	-20, 0
	defect	0, -20	-10, -10

Fig. 1. Payoff Matrix for Emma and John's choices:

Referring to Fig.1. defect = confess and cooperate = don't confess [2].

In this project, we specifically investigate the Iterated Prisoner's Dilemma (IPD) where we use and compare various Local Search and Optimization algorithms such as Hill-climbing, Simulated Annealing, Tabu Search, and the Genetic Algorithm.

The IPD is more useful for modeling real world situations where interactions are often repeated and decisions have long-term consequences. The iterated prisoner's dilemma allows for player strategies such as: All-Cooperate: Always cooperate every round. All-Defect: Always defect every round. Tit for Tat: Start by cooperating, and then mimic the opponents previous decision. In the Methodology section, we will introduce even more strategies for the IPD.

2. Literature review

A. The Iterated Prisoner's Dilemma and Its Strategic Importance

In several fields, including game theory, psychology, and artificial intelligence, the IPD has long been a key model for researching cooperative behavior, conflict, and strategy. The IPD expands the single-use game into a recurring interaction, allowing players to modify their tactics in response to their opponent's actions. It is based on the classic dilemma first presented by Flood and Dresher (1950) [4]. This iterative process draws attention to the paradox that, although defection might benefit one individual more immediately, long-term collaboration frequently produces more advantageous results for both parties. In the past, researchers—most notably Robert

Axelrod—have stressed that effective IPD techniques need to be "nice," forgiving, non-envious, and vindictive when needed [5].

B. Evolutionary Computation in IPD: Genetic Algorithms

The research done by Collins. (2019) investigates the use of genetic algorithms (GAs) to generate strategies for the problem, building on the findings of IPD research. Potential strategies are encoded by genetic algorithms as chromosomes that reproduce, mutate, and crossover, simulating natural selection processes [6]. Because it enables a thorough investigation of the strategy space and may yield new methods that outperform traditional strategies like cooperator, defector, and Tit-for-Tat, this method is especially well-suited to optimization problems like the IPD. Previous research in this field, such as studies by Carr (2015), has shown how promising evolutionary techniques are for producing high-performing and adaptive solutions.

C. The Role of the Moran Process and Population Dynamics

The literature also emphasizes how crucial population-based models like the Moran process in relation to the development of IPD tactics. The process works by assigning the score of the individual to a fitness level, then a greater fitness level determines the a greater probability of reproduction, showing how the Moran process mimics natural selection [3] [7]. In order to maintain genetic diversity and avoid premature convergence to local optima, a mutation rate is incorporated into the process. Making sure that the mutation rate remains low is an important factor in this process [3]. The interaction of reproduction, mutation, and population may cause bottlenecks which restrict the number of strategies that advance to the following generation. Therefore the Moran process is important for promoting innovation and resulting in the formation of high-scoring strategies, as noted by Collins. (2019).

D. Finite State Machines as Strategy Encoders

The Collins paper made a large contribution by representing IPD techniques using Finite State Machines (FSMs). FSMs work by defining an initial state and final state, and then have arrows called transition states that can lead to different states and eventually reach the final state [9] (see figure 2 that depicts the initial state "1" that is also the final state. State 1 can transition between D/D and/or C/C i.e., defect and/or cooperate respectively). FSMs provide a more adaptable framework that can encapsulate intricate state transitions than basic lookup tables, which simply take into account the recent past move [3]. Because each FSM is made up of a collection of states and actions, the strategy can be adjusted dynamically in response to the opponent's past movements [3]. In addition to streamlining the decision-making process, this approach offers a way to capture complex strategies like conditional retaliation and forgiveness. The study demonstrates how FSM-based techniques can capture subtle behavior that is on par with or better than traditional methods, even when there are only a few states.

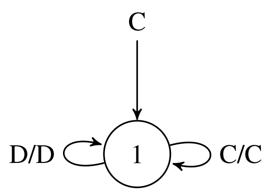


Fig. 2. Finite state machine representing the "Tit-for-Tat" strategy [3].

E. Empirical Findings and Comparative Performance

Through systematic experimentation, the study demonstrates that finite state machines evolved via genetic algorithms can produce strategies that outperform standard approaches in IPD tournaments. For instance, evolved FSM strategies achieved higher average scores than classical strategies such as Tit-for-Tat, defector, and cooperator. This finding reinforces the idea that incorporating evolutionary dynamics can lead to the discovery of strategies that balance cooperative and competitive behaviors in more effective ways. Moreover, the experiments show that increasing the number of generations in the genetic algorithm yields incremental performance improvements, highlighting the benefits of prolonged evolutionary search.

3. Methodology

In this study, we aim to evolve effective strategies for the iterated Prisoner's Dilemma by comparing four optimization techniques: Genetic Algorithm (GA), Hill Climbing (HC), Tabu Search (TS), and Simulated Annealing (SA). The objective is to maximize the total fitness of strategies over multiple rounds of the game. Here, fitness is defined as the total payoff accumulated by a strategy when it is played against a set of opponents. In our simulations, the fitness function [15] is given by equation (1):

$$F(s) = \sum_{i=1}^{R} p_i(s) \tag{1}$$

where $p_i(s)$ is the payoff in round i for strategy s, and R is the total number of rounds. In simpler terms, we are summing the payoffs from each round to determine how well a strategy performs overall.

A. ***Strategies***

We consider seven distinct strategies:

1) **Always Cooperates (ALLC)** This strategy consistently chooses to cooperate in every round, regardless of the opponent's actions.
2) **Always Defects (ALLD)** This strategy always defects, aiming to maximize immediate gains without considering future retaliation.
3) **Tit-for-Tat (TFT)** This strategy starts with cooperation and then mirrors the opponent's previous move in the previous round.
4) **Tit-for-n-Tat (TFnT)** A variation of TFT that tolerates up to n defections before retaliating with defection.

5) **Suspicious Tit-for-Tat (STFT)** Similar to TFT, but begins with defection instead of cooperation, assuming the opponent may be untrustworthy.
6) **Grim Trigger (GRIM)** Cooperates until the opponent defects once, after which it defects permanently as punishment.
7) **Win-Stay, Lose-Switch (WSLS)** Repeats its previous action if the payoff is high; otherwise, it switches. This strategy is also known as *Pavlov*.

B. *Optimization Methods*

C. *Genetic Algorithm (GA)*

GA mimics natural selection, evolving a population of solutions over generations to improve performance. Each solution is represented as a "genome" (e.g., a vector or binary string). GA uses three main operators: selection, crossover, and mutation.

Selection: Strategies are chosen as parents based on fitness. In *roulette wheel selection*, the probability of selecting strategy s_i is proportional to its fitness:

$$P(s_i) = \frac{F(s_i)}{\sum_{j=1}^{N} F(s_j)}$$

Crossover: Two parent strategies combine parts of their genomes to create offspring. For one-point crossover, a random index k is chosen, and the offspring's genome is formed by combining parts from both parents:

$$o = [s_i(1:k) \parallel s_j(k+1:L)]$$

Mutation: A small random change is introduced into the offspring's genome with a probability μ. This prevents the algorithm from getting stuck in local optima.

GA continues for a set number of generations or until no significant improvement is observed, making it effective for large, complex search spaces.

D. *Hill Climbing (HC)*

Hill Climbing is a simple iterative local search algorithm.It starts with an initial solution and then makes small adjustments to find an improved solution [12]. The process is as follows, equation (4):

$$s' = s + \delta \tag{2}$$

where δ is a small change applied to the current solution s. If the new solution s' has a higher fitness (i.e., $F(s') > F(s)$), it replaces the current solution. This is repeated until no neighboring solution offers an improvement. Although HC is an efficient method, it can get stuck in local optima, which is a major drawback. This is due to the fact that it may find the best solution in its immediate area, but not the best overall in the search space.

E. *Tabu Search (TS)*

Tabu Search is an advanced local search method that is an extension of local search methods like Hill Climbing. It uses a special memory structure called the tabu list, which keeps track of recently explored solutions or moves. This prevents the algorithm from revisiting the same solutions and getting stuck in cycles [14].

$$s' = \arg\max_{s_j \notin T} F(s_j) \tag{3}$$

where T is the tabu list.

Even if a move is marked as tabu, an aspiration criterion may allow it if it leads to a solution better than any previously found (denoted as F_{best}) . The tabu list has a fixed size, meaning older entries are eventually removed, allowing the search to explore new areas. TS is beneficial in complex search spaces where local optima are common, as it encourages broader exploration.

F. *Simulated Annealing (SA)*

Simulated Annealing is an optimization technique inspired by the annealing process in metallurgy. It allows the acceptance of worse solutions with a certain probability, enabling the algorithm to escape local optima. At each iteration, a neighboring solution$s\prime$ is generated, and the change in fitness is calculated [11]:

$$\Delta F = F(s') - F(s) \tag{4}$$

If the new solution is better ($\Delta F \geq 0$), it is accepted. If not, it is accepted with a probability given by the Boltzmann Distribution [1], [11]:

$$P(\text{accept}) = e^{\frac{\Delta E}{T}} \tag{5}$$

where T is the current temperature, a parameter that controls the likelihood of accepting worse solutions. Over time, the temperature is gradually decreased following a cooling schedule:

$$T_{k+1} = \alpha T_k \tag{6}$$

with α (typically between 0.90 and 0.99) being the cooling rate. This schedule reduces the probability of accepting poor solutions, allowing the algorithm to focus on refining the best-found solution. SA is particularly effective for problems with complex landscapes, where many local optima could trap simpler optimization methods.

G. *Evaluation Metrics*

Fitness Score: Measures the total payoff accumulated over multiple rounds against predefined opponent strategies. A higher fitness score indicates a better-performing strategy. Each optimization algorithm uses fitness differently: GA determines which strategies survive and evolve, HC continuously adjusts strategies to maximize fitness, TS uses fitness to explore new solutions while avoiding previously visited ones, and SA balances exploration (accepting poor solutions) and exploitation (improving the best solution). Comparing fitness scores across algorithms helps identify the most effective strategy evolution method.

Execution Time: Measures how long an algorithm takes to find an optimized strategy. Faster algorithms may converge quickly but risk local optima, while slower algorithms conduct a broader search, potentially leading to better solutions. By analyzing both metrics, the optimal balance between solution quality and computational efficiency can be determined.

4. Experimental Setup

In our experiments, each optimization technique is evaluated in a controlled simulation of the iterated Prisoner's Dilemma.

A. *Hyperparameter Fine-tuning*

In all of the optimization methods, we employ grid search based approach to find the best hyperparameter configuration.

1) Genetic Algorithm (GA): The GA evolves a population of candidate solutions over successive generations using selection, crossover, and mutation. The key hyperparameters used in our experiments are:

- Population Sizes: 50, 100, 150
- Mutation Rates: 0.0001, 0.01, 0.05, 0.1, 0.5
- Crossover Rates: 0.5, 0.7, 0.9
- Memory Depths: 2, 3, 4
- Generations: 100
- Game Rounds per Evaluation: 500

The fitness of each strategy is measured by stimulating 500 rounds against predetermined competitor strategies.

2) Hill Climbing (HC): Hill Climbing performs local search from an initial solution by applying small adjustments to improve it iteratively. The hyperparameters used for HC are:

- Initial Solutions (Population Sizes): 20, 50, 100
- Iterations: 50, 100, 150
- Memory Depths: 2, 3, 4
- Game Rounds per Evaluation: 500

Each starting solution is gradually improved through local modifications until no further enhancements are observed.

3) Tabu Search (TS): The tabu search algorithm improves local search by maintaining a tabu list that records recently visited solutions, thereby preventing cycles. The key hyperparameters include:

- Initial Solutions (Population Sizes): 20, 50, 100
- Iterations: 50, 100, 150
- Tabu List Sizes: 5, 10, 15
- Memory Depths: 2, 3, 4
- Neighbor Generation: 10 moves per iteration
- Aspiration Criterion: Enabled
- Game Rounds per Evaluation: 500

5. Results

In this section, we present the main numerical outcomes and discuss the figures generated during the experiments.

Figures

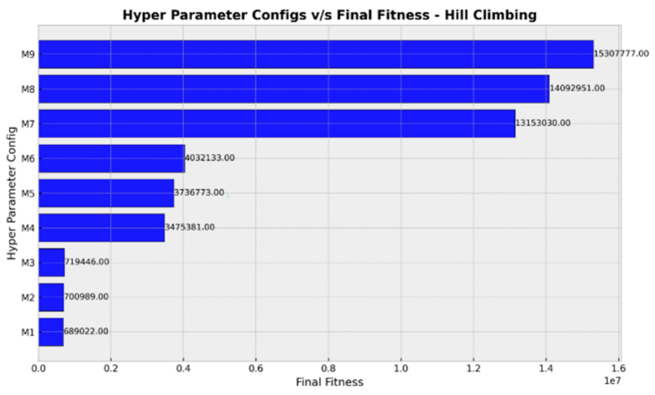


Fig. 3. Hyperparameter Configs vs. Final Fitness – Hill Climbing

Hill Climbing Analysis: Figure 3 shows the final fitness values for HC across different hyperparameter configurations. There is a clear progression in fitness from M1 to M9. The highest fitness is achieved by M9 (Population=100, Iterations=150, Memory Depth=3) at approx. 1.53×10^7. Notably, smaller populations or fewer iterations result in lower maximum fitness values (e.g., M1 at 6.89×10^5). Thus, increasing both population size and iteration count yields a more thorough local search and a higher payoff.

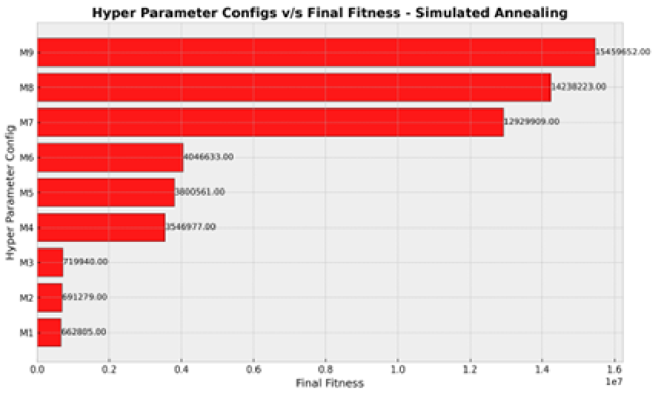


Fig. 4. Hyperparameter Configs vs. Final Fitness – Simulated Annealing

Simulated Annealing Analysis: In Figure 4, M9 (Population=100, Iterations=150, Temp=50, Cooling=0.95, Memory Depth=2) achieves the highest fitness at approx. 1.55×10^7. The results confirm that a balanced cooling schedule and sufficient iterations allow SA to escape local optima and match (or closely approach) HC's performance.

Tabu Search Analysis: According to Figure 5, the best TS configuration (M9: Population=100, Iterations=150, Tabu Size=5, Memory Depth=4) achieves a final fitness of approx. 1.33×10^7. This is slightly lower than the best results from

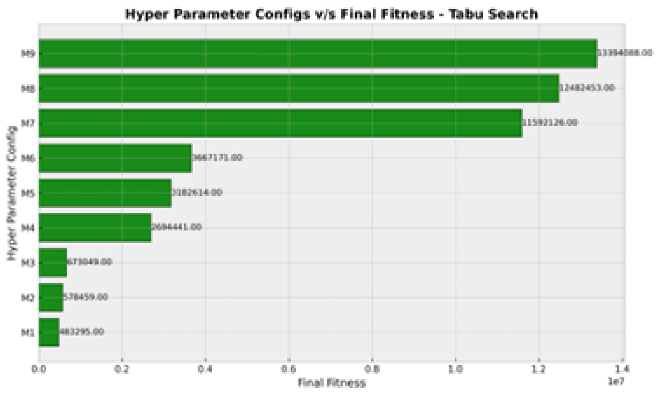


Fig. 5. Hyperparameter Configs vs. Final Fitness – Tabu Search

HC and SA, suggesting that while TS avoids local optima, it may not explore as thoroughly as SA, or run as efficiently as HC under these hyperparameters.

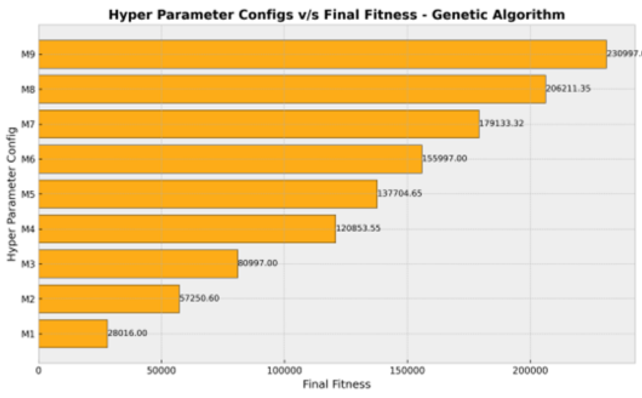


Fig. 6. Hyperparameter Configs vs. Final Fitness – Genetic Algorithm

Genetic Algorithm Analysis: Figure 6 demonstrates that GA's best fitness (M9: Population=150, Mutation=0.0001, Crossover=0.9, Memory Depth=3) is 2.31×10^5, which is significantly lower than the best results from HC, SA, and TS. However, GA also displays the greatest variability. The high execution time (see Table 4) may indicate overhead in evolving large populations over multiple generations. However, by increasing the population size, we saw diminishing returns.

Best Strategies Evolution per Model: Figure 7 illustrates how the composition of strategies changes over time in the best-performing run of each method. As shown in Figure 7, SA and HC exhibit a similar pattern, gradually favoring Always Cooperate in later generations. In contrast, TS shows minor fluctuations between strategies but remains relatively stable overall. In GA, Always Defect and STFT decline early in the evolution, leading to a final distribution largely dominated by Always Cooperate.

Cross-Model Fitness Evolution: In Figure 8, SA and HC are the top-performing methods, with SA slightly outperforming HC in the final stages. Each method achieves a final average fitness score of 1.53×10^7 and 1.55×10^7, respectively. Their lines intersect around a generation value of 25 and an average fitness of approx. 1.3×10^7. Then, they separate, demonstrating their competitive nature, with both methods alternating in performance during different stages of the optimization. TS starts with a moderate fitness and shows consistent improvement over iterations. It achieves a

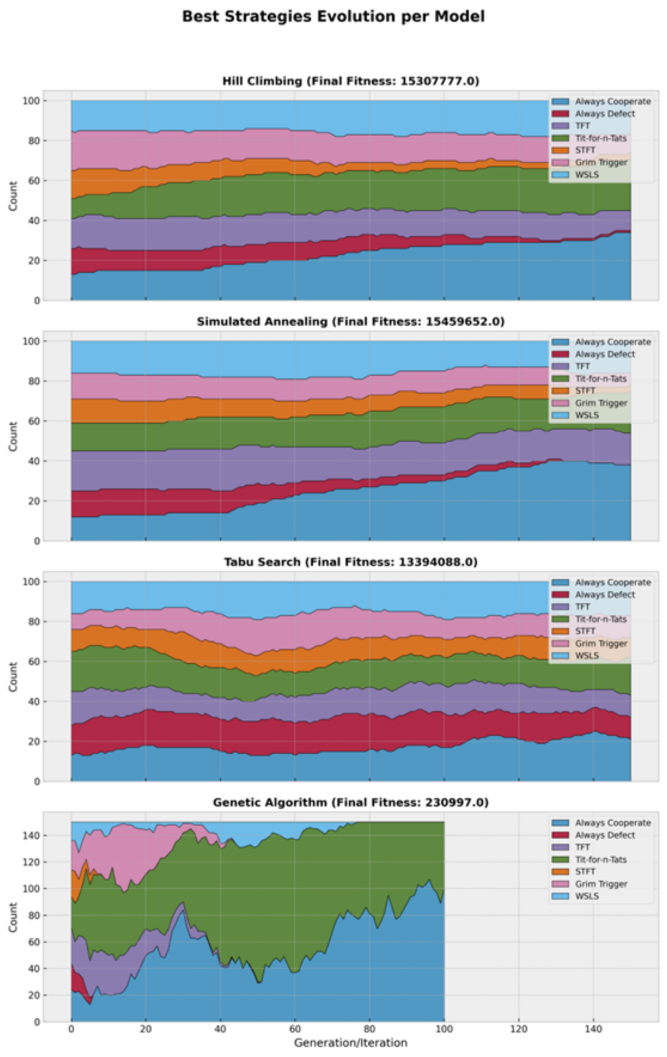


Fig. 7. Best Strategies Evolution per Model

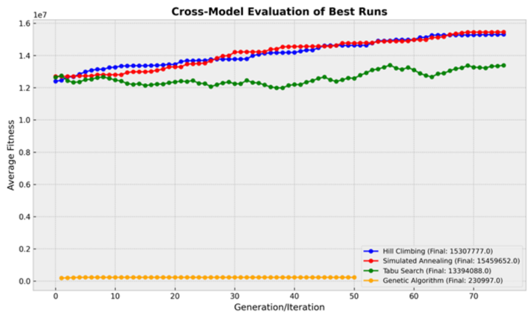


Fig. 8. Cross-Model Evaluation of Best Runs

final average fitness of 1.34×10^7, which is lower than HC and SA. The GA performs significantly worse than all other methods. It's fitness score remains constant across all generations (1.34×10^7) and displays slow improvement.

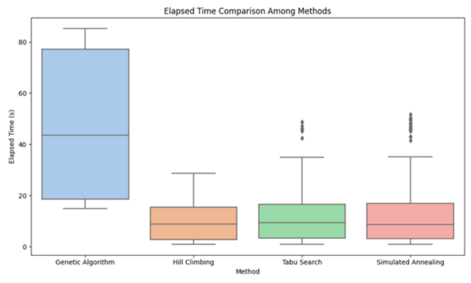


Fig. 9. Elapsed Time Comparison Among Methods

Execution Time Analysis: In Figure 9, we can observe that GA exhibits the highest median elapsed time (approx. 43s), indicating that it is the most computationally expensive method. The interquartile range (IQR) is relatively wide, suggesting variability in execution time across different runs. In comparison, HC has the lowest median elapsed time (approx. 10s), making it the fastest method among the four. This efficiency makes HC suitable for scenarios where quick results are needed. Although GA is more time consuming, it explores a larger search space and is more likely to find globally optimal solutions. TS and SA offer a balance between efficiency and exploration, making them suitable for problems where both speed and solution quality are important.

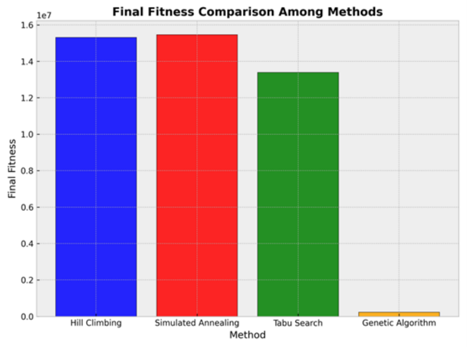


Fig. 10. Final Fitness Comparison Among Methods

Final Fitness Comparison: Figure 10 demonstrates that HC and SA both surpass 1.5×10^7 in final fitness, making them the most effective at maximizing fitness in this experiment. TS reaches a final fitness of approx. 1.33×10^7, which may result from the constraints imposed by the tabu list, limiting search space exploration. GA is substantially lower at 2.3×10^5, likely due to its higher computational complexity and slower convergence rate.

Strategy Count Analysis: In Figure 11, the GA ends with the highest average count of TF-n-Tats (approx. 24.1), followed by TFT (approx. 22.1). The presence of Always Cooperate (approx. 18.2) is also relatively high, indicating that cooperative strategies are favored in an evolutionary setting. Always Defect (approx. 4.8) and STFT (approx. 4.5) appear in lower numbers, suggesting that purely exploitative or overly forgiving strategies are not as evolutionarily stable. The high presence of TFT and TF-n-Tats shows that strategies based on reciprocity and conditional cooperation do well over time, likely because they can handle different types of opponents effectively. Always Cooperate is the dominant strategy in HC (approx. 19.2) as well as SA (approx. 18.8). HC and SA favor a balanced mix of cooperative and retaliatory strategies (Always Cooperate, TF-n-Tats, WSLS).

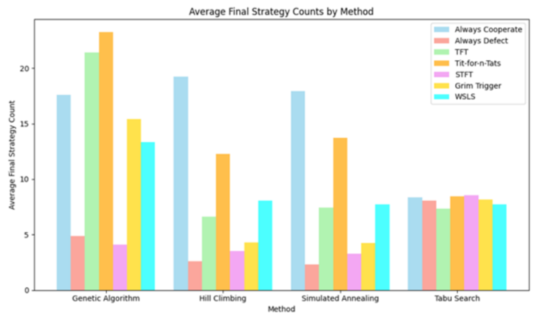


Fig. 11. Average Final Strategy Counts by Method

In TS, the strategies are more evenly distributed compared to the other methods. In other words, all strategies have a similar count (ranging from approx. 7.0 to 9.0). STFT has the highest representation (approx. 9.0) among the other strategies, suggesting that TS favors exploitative and short-term forgiving strategies. The relatively balanced distribution indicates that TS does not heavily favor any particular strategy, due to its ability to escape local optima and cycle through different strategy sets.

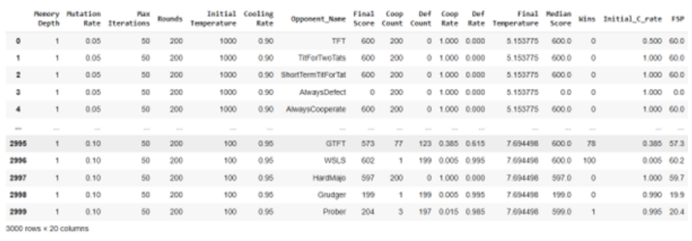
A. *Case study on Machine Learning*

Part-1 Supervised Learning

We aim to use machine learning models to classify whether a strategy generated by search algorithms such as Hill Climbing, Tabu Search, Simulated Annealing, and Genetic Algorithm is "good" or "bad". To generate the training datasets, we simulate iterated prisoner's dilemma games between machine-generated strategies and human-designed strategies. In our simulations, player1 employs the search-based strategy, while player2 use one of fifteen human-designed strategies (TFT, Tit-ForTwoTats, ShortTermTitForTat, AlwaysDefect, AlwaysCooperate, RandomStrategy, GrimTrigger, ZDGTFT2, Extort2, HardJoss, GTFT, WSLS, HardMajo, Grudger, Prober[10]).

The simulation was run for 50 iterations for each machine-generated strategy, during which various algorithm parameters were varied. Specifically: 1. In the SA dataset, we varied the Memory Depth, Mutation Rate, Initial Temperature, Cooling Rate, and Final Temperature. 2. In the HC dataset, we adjusted the Memory Depth and Mutation Rate. 3. In the TS dataset, we modified the Memory Depth, Mutation Rate, and Tabu Tenure. 4. In the GA dataset, we changed the number of Generations, the Crossover Rate, and the Mutation Rate. This systematic parameter variation allowed us to generate data with a wide range of algorithm behaviors across different search methods.

Then, we combined the resulting data from all simulations, removed redundant columns, and formed a unified dataset for machine learning. Our training dataset contains 10,501 in-



	Memory Depth	Mutation Rate	Max Iterations	Rounds	Initial Temperature	Cooling Rate	Opponent_Name	Final Score	Coop Count	Def Count	Coop Rate	Def Rate	Final Temperature	Median Score	Wins	Initial_C_rate	FSP
0	1	0.05	50	200	1000	0.90	TFT	600	200	0	1.000	0.000	5.153775	600.0	0	0.500	60.0
1	1	0.05	50	200	1000	0.90	TitForTwoTats	600	200	0	1.000	0.000	5.153775	600.0	0	1.000	60.0
2	1	0.05	50	200	1000	0.90	ShortTermTitForTat	600	200	0	1.000	0.000	5.153775	600.0	0	1.000	60.0
3	1	0.05	50	200	1000	0.90	AlwaysDefect	0	200	0	1.000	0.000	5.153775	0.0	0	1.000	0.0
4	1	0.05	50	200	1000	0.90	AlwaysCooperate	600	200	0	1.000	0.000	5.153775	600.0	0	1.000	60.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2995	1	0.10	50	200	100	0.95	GTFT	573	77	123	0.385	0.615	7.694498	600.0	78	0.385	57.3
2996	1	0.10	50	200	100	0.95	WSLS	602	1	199	0.005	0.995	7.694498	600.0	100	0.005	60.2
2997	1	0.10	50	200	100	0.95	HardMajo	597	200	0	1.000	0.000	7.694498	597.0	0	1.000	59.7
2998	1	0.10	50	200	100	0.95	Grudger	199	1	199	0.005	0.995	7.694498	199.0	0	0.990	19.9
2999	1	0.10	50	200	100	0.95	Prober	204	3	197	0.015	0.985	7.694498	599.0	1	0.995	20.4

3000 rows × 20 columns

Fig. 12. Unified dataset for machine learning

stances and 6 features, with the "Evaluation" attribute serving as the target label. A strategy is labeled 1 ("good") if, in a given round, it obtains at least 60% of the total score and maintains a cooperation rate of 0.6 or higher; otherwise, it is labeled 0 ("bad").

For prediction, each instance includes the features "Algorithm Used", "Memory Depth", "Mutation Rate", "Opponent Name", and "Coop Rate" and the model outputs the predicted Evaluation (1 or 0). We tested our trained models with 10 sample inputs.

First, we can see from the table that any strategy with a coop rate below 0.6 will evaluated as bad strategy by all the models.Second, we noticed that most of the strategies are evaluated as bad strategy.Finally, we see that in some cases, different models produced conflicting predictions (e.g., some predicted 1 while others predicted 0 for the same input).

Next, let's compare the four models by the accuracy it predicted on the testing set. We can see from the table

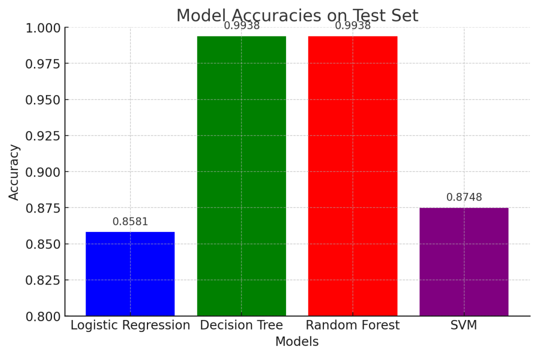


Fig. 13. Prediction accuracy by 4 machine learning models on test set

that Decision Tree and Random Forest achieved the highest performance, each with approximately 99% accuracy. In contrast, Logistic Regression and Support Vector Machine yielded accuracies below 90%.

Based on our experiments training machine learning models to classify machine-generated strategies for the Iterated Prisoner's Dilemma, we can draw several conclusions: 1. The majority of machine-generated strategies tend to be classified as bad strategy. 2. Decision Tree and Random Forest models consistently outperform Logistic Regression and Support Vector Machine models in terms of prediction accuracy.

However, our study also revealed two key challenges in the evaluation process: 1. Opponent Dependence: The performance evaluation is based on the total score achieved in a round, but this score is highly dependent on the opponent's behavior. For example, if the opponent is AlwaysDefect, the maximum score attainable in a 200-round game is only 200. Consequently, when facing AlwaysDefect, even well-designed strategies may be unfairly labeled as "bad" by the model. 2. Cooperation Rate Threshold: The models were trained to label a strategy as "good" only if its cooperation rate was at least 0.6. This means that any strategy with a cooperation rate below 0.6 is classified as "bad", even though a strategy with a high defection rate might achieve nearly full score when competing against opponents like AlwaysCooperate.

To overcome these issues, it will be necessary to develop more robust evaluation methods. Addressing these challenges presents a promising avenue for future research.

For future work, we plan to create more robust and balanced training datasets to improve model reliability. Additionally, we will explore a wider range of machine learning techniques, including neural networks, to gain deeper insights into the Iterated Prisoner's Dilemma.

Part-2 Reinforcement Learning

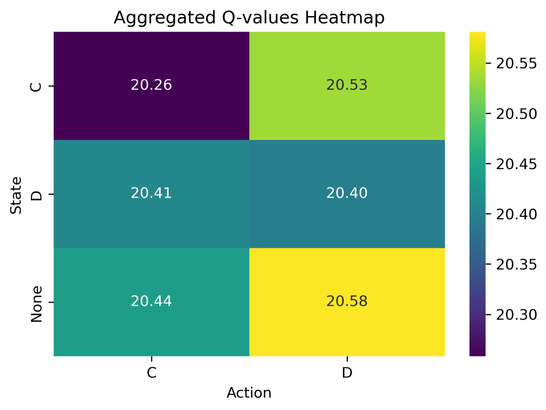


Fig. 14. Heatmap showing learned policy

Aggregated Analysis: The overall aggregated heatmap shows the average Q-values across the entire dataset. Formula:

$$Q_{\text{mean}}(s,a) = \frac{1}{N_{s,a}} \sum_{i=1}^{N_{s,a}} Q_i$$

Evolution Analysis: The data is split into bins (here, every 10 rows). For each bin, the mean Q-value is computed, and then a rolling average (window = 100) smooths these values. Formula for the rolling (moving) average:

$$S_i = \frac{1}{w} \sum_{j=i-w+1}^{i} Q_j$$

Best Strategy Determination: At each evolution step (each bin), the strategy (State, Action) with the highest smoothed Q-value is selected:

$$(s^*, a^*) = \arg\max_{(s,a)} S_i(s, a)$$

Overall, the aggregated Q-values suggest that for states "C" and "None," defecting yields a slightly higher average Q-value, whereas for state "D," cooperating is marginally better. However, the differences are small, indicating that neither action strongly dominates across all states.

6. Discussion

The chosen hyperparameter ranges are based on preliminary experiments and standard practices in the literature. In the GA, larger populations can improve diversity (150 as seen in Table 4 of Appendix), while higher mutation rates promote exploration. For HC and TS, the number of iterations is critical to ensuring that local optima are adequately explored; however, HC may still get trapped if the landscape is complex. In TS, the tabu list and aspiration criterion help mitigate this issue by encouraging broader exploration. In SA, the cooling rate α and the initial temperature T_0 are crucial for balancing exploration and exploitation. A cooling schedule that is too rapid may lead to premature convergence, while a very slow cooling process increases computational time. This systematic variation allows us to analyze performance trade-offs in terms of final fitness, convergence speed, and the diversity of strategies evolved.

HC and SA achieve the highest fitness (1.35×10^7 and 1.29×10^7, respectively) while also maintaining efficient runtimes, with HC being the fastest (approx. 29s). Tabu Search performs well but falls slightly behind, and the Genetic Algorithm struggles with lower fitness (2.3×10^5) and longer runtimes (approx. 80s). Strategies that balance cooperation and retaliation, like TFT and WSLS, tend to be more successful, whereas purely cooperative or defecting strategies are less stable. Overall, local search methods prove more effective in this setup, while the Genetic Algorithm may require further tuning to compete.

7. Conclusion

In this paper, we investigated the IPD using four main optimization algorithms— HC, SA, TS, and the GA—alongside a machine learning case study. Our experimental results indicate that HC and SA generally yield the highest final fitness, while TS offers competitive performance but falls slightly behind. GA exhibits the largest variance in outcomes and lower final fitness, partly due to its higher computational overhead and slower convergence.

From a strategic standpoint, reciprocating approaches (e.g., Tit-for-Tat and its variants) and conditionally cooperative strategies (e.g., Win-Stay, Lose-Switch) tend to dominate over purely cooperative or purely defecting strategies. In the machine learning component, we found that Decision Tree and Random Forest models classify evolved strategies with higher accuracy (99%) than Logistic Regression or SVM. However, certain evaluation biases—such as opponent dependence and a fixed cooperation-rate threshold—can unfairly label otherwise strong strategies as "bad".

Overall, our findings suggest that local search methods (HC and SA) strike a good balance between solution quality and execution time in the IPD setting. Future work could explore advanced deep reinforcement learning methods (e.g., DQN variants) and refined evaluation metrics to handle a wider variety of opponent behaviors and strategic nuances.

8. Acknowledgments

A. Extra Materials

We acknowledge the use of ChatGPT for assisting in idea generation and content refinement.

B. Student Contributions

Below is a summary of each student's contributions to the project:

- **Matt Davies:** Conducted theoretical research, introduction, literature review, and documented references.
- **Jolin Lodhiya:** Developed the experimental setup, analyzed the results, and created visualizations.
- **Sara Sarhill:** Implemented optimization algorithms, hyper parameter exploration and fine tuning, incorporated strategies and evaluated results.
- **Zhuqing Sha:** Generated the training set and did the case study of supervised machine learning.

References

[1] S.J. Russell, P. Norvig, "Artificial Intelligence A Modern Approach Fourth Edition Global Edition", Pearson Education Limited, 2022.

[2] R. Gras, "Project", Artificial Intelligence Concepts, University of Windsor, 2025, pp.1-2.

[3] D. R. Collins, "Review: A Genetic Algorithm Approach to the Iterated Prisoner's Dilemma", Dept. of Computer Science, University of Windsor, 2019.

[4] S. Kuhn, "Prisoner's Dilemma," Stanford Encyclopedia of Philosophy, Stanford University, 2014.

[5] R. Axelrod, "The evolution of cooperation," New York: Basic Books, 2006.

[6] J. Carr, "An Introduction to Genetic Algorithms," Whitman.edu, 2014.

[7] P.A.P. Moran, "Random Processes in Genetics". Mathematical Proceedings of the Cambridge Philosophical Society, Volume 54, Issue 1, 1958.

[8] J. Garcia, "Advantages & Disadvantages of Finite State Machines: Switch-cases, C/C++ Pointers & Lookup Tables (Part II)". IOT FUNDAMENTALS, ubidots.com, 2018.

[9] itemis, "What is a state machine?". itemis CREATE, itemis.com, 2025.

[10] V. Knight, "Axelrod's first tournament". Tournaments, Vincent Knight Revision, axelrod.readthedocs.io, 2015.

[11] Van Laarhoven, Peter JM, et al. "Simulated annealing." Springer Netherlands, 1987.

[12] Selman, Bart, and Carla P. Gomes. "Hill-climbing search." Encyclopedia of cognitive science 81.333-335 (2006): 10.

[13] Mathew, Tom V. "Genetic algorithm." Report submitted at IIT Bombay 53 (2012): 18-19.

[14] Gendreau, Michel, and Jean-Yves Potvin. "Tabu search." Search methodologies: introductory tutorials in optimization and decision support techniques (2005): 165-186.

[15] Harman, Mark, and John Clark. "Metrics are fitness functions too." 10th International Symposium on Software Metrics, 2004. Proceedings.. Ieee, 2004.