(Re)discovering Effective Strategies in the Iterated Prisoner's Dilemma by Means of an Evolutionary Algorithm

Stijn Boosman *s1018438*
Florian Handke *s1017134*
Mart van 't Spijker *s1018582*
Radboud University *June 23, 2023*

Github

1 Introduction

The Prisoner's Dilemma (PD) is a widely recognized tool in game theory that is often utilized to examine the dynamics of cooperation, conflict resolution, and competition within systems comprising autonomous agents [1, 3, 8]. In an abstract sense, it represents a one-time interaction between two entities that are confronted with the choice of either cooperating or competing with one another while lacking the knowledge of the adversaries' choice. In the absence of external incentives, the classical PD yields a system state that is sub-optimal when viewed from a social welfare perspective. That is, both entities engage in competition, leading to a lower aggregated reward than could have been attained when mutually cooperating [7]. Unfortunately, once this system state has been entered, neither entity has the incentive to switch from competition to cooperation as such a transition would yield costs when not carried out in conformity. However, when interaction extends beyond a one-time encounter, within an environment comprised of more than just two entities, the dynamic changes. A strategy's success is then not only measured relative to the opposite but relative to all entities in the system. While competition still remains a valid option, it can now face potential repercussions in successive rounds of the PD. As a result, the aforementioned sub-optimum is not upheld globally as it can be out-competed by cooperative strategies in the long run. Yet, unconditional cooperation is seldom a viable approach, for it can be exploited by partially defective entities. Consequently, for the system to transition to a cooperative state and remain there, more sophisticated strategies are required. A first attempt in finding such strategies was made by Axelrod in 1980 who conducted a tournament with strategies submitted by researchers from various domains [1, 2]. The strategy that was found to be most effective is nowadays known as *Tit-for-Tat* and is considered a benchmark for what are claimed to be dominant strategies. In the decades thereafter, others have devised methods for finding effective strategies, either by means of *Round-Robin* based competitions where strategies are pitted against each other (Axelrod's tournament), or using evolutionary competitions that select strategies on individual fitness or multi-objective fitness functions. To employ the latter method, players were assigned a genetic structure. Several options were proposed for this purpose, including look-up tables, and finite-state machines [6].

This project aims to explore the space of strategies by means of evolutionary competition using neural networks as the genetic structure of players. As the validity of our approach can be measured by its ability to find effective strategies, including well-established ones such as *Tit-for-Tat*, we will address the following research question:

To what extent can an evolutionary algorithm using neural network players find effective strategies in the iterated Prisoner's Dilemma?

Before we will provide a more detailed account of the methods used to investigate this question, we will give a concise overview of the Prisoner's Dilemma and its associated terminology. Additionally, we will summarize the findings from related previous research. Concluding this report, we will present our results and discuss their limitations.

1.1 The Prisoner's Dilemma

As outlined before, the PD is a two-player game where the individual players have to decide between cooperating or competing with one another without the ability to communicate beforehand. In the classical setup, the game is only played once. Each player chooses to either cooperate or defect and is subsequently rewarded according to a payout matrix, such as the one in table 1.

P2 / **P1**	**Coop.**	**Defect**
Coop.	+5	+8 / 0
Defect	0 / +8	+2

Table 1: A possible payoff matrix of the PD

The matrix reveals that whatever the adversary chooses to do, it is beneficial to defect. In game theory, this defines a *dominant strategy.* In a scenario where both players choose to play this strategy, the resulting game state is considered a *Nash equilibrium* which is defined as a state in which neither player has the incentive to switch actions. Unfortunately, the Nash equilibrium in the classical PD is *Pareto inefficient*, i.e. a different game state would yield a higher overall result, namely when both players cooperate. However, as soon as there are more than just two players playing the PD in parallel, the return of a player is no longer solely measured in relation to their direct opponent but also relative to the other players in the population. Hypothetically, cooperative players can now out-compete defective players as long as they do not play against too many of them. But in a scenario where every player can play against any other player, defecting will still dominate cooperation as players interact on a one-time basis. Hence, they cannot recognize defective players beforehand and adapt their strategies accordingly. The resulting game state will again be Pareto inefficient. Yet, by allowing the same two players to play against each other more than just once, the dynamics change. Cooperative players can now observe what their opponent will do and adaptively defect if needed. Two examples of cooperative strategies that use retaliation are shown in *table* 2.

Strategy	**1**	**2**	**3**	**4**	**5**	...
Hawk	D	D	D	D	D	D
Grim Trigger	C	D	D	D	D	D
Tit-For-Tat	C	D	D	D	D	D

Table 2: Round-based outcomes of retaliative cooperative strategies (grim trigger, tit-for-tat) vs. a permanent defector (hawk)

Both, the *Grim Trigger* as well as the *Tit-for-Tat* strategy retaliates upon observing defection. Although not apparent in this example, *Tit-for-Tat* does not retaliate forever while the *Grim Trigger* strategy defects all rounds consecutive to observing defection. Due to the existence of retaliative strategies, permanent defection is no longer an unconditionally dominant strategy. In fact, there are no unconditional dominant strategies anymore, as the most effective strategy depends on the population of strategies. Accordingly, there might be populations with near Pareto efficient Nash equilibria.

2 Related Research

In 1980, Axelrod [1], [2] invited experts from various fields to design and submit agents to participate in two rounds of an iterated prisoner's dilemma tournament. The first round was meant as a primer, to get the participants familiar with the facets of the game, while the second round provided candidates the opportunity to improve their design. Tit-for-tat, a then-unknown strategy that was specifically created for this tournament, emerged victorious in both rounds. Axelrod's tournament marked the first time a competition like this had been published. Since then, more research and competition have followed. The goal of these publications was usually to find better strategies, to find an algorithm that can invent better strategies or both. Recently, Harper [4] compared many different strategies that have been devised over the years, including those in Axelrod's library. The paper looks at the various ways agents can be designed in order to be used in reinforcement learning (such as evolutionary algorithms). They found that agents were often represented as either finite state machines, lookup tables or artificial neural networks. When pitted against each other, the best-performing strategy was based on a lookup table, but both finite state machine and neural network-based agents achieved several top 10 positions, indicating that all methods have enough potential to justify further research.

3 Methods

In order to assess the effectiveness of various versions of our algorithm, we opted to evaluate them based on their ability to generate effective strategies; specifically, players that exhibited evolutionary robustness within the framework of evolutionary competitions. To accomplish this, we employed three key components: Players, an optimization procedure, and an algorithm designed to identify a player's strategy. The content of this section adheres to the order in which the components were mentioned.

3.1 The Players

In the iterated Prionser's dilemma (IPD), a player must possess both, the ability to act and the capability to reason. In this context, acting refers to the generation of responses, cooperation (C), and defection (D), upon observing the opponent's previous actions. Reasoning, on the other hand, is the transformation by which these observations are mapped into responses. While acting as such is a simple transformation from the reasoning outcome to the domain $\{C, D\}$, reasoning in the IPD can be highly complex, encompassing stochastic and self-referential elements. Considering the scope of this project, we chose to constrain players to deterministic reasoning by means of the perhaps simplest system, *Boolean algebra*. Even within this system, the complexity of a strategy grows exponentially with a player's processing capacity [1]. In order to manage this complexity, we have imposed a restriction on our players, allowing them a maximum processing capacity of two actions, resulting in 128 unique strategies. For comparison, increasing the maximum processing capacity by only a single bit would already yield 32768 unique strategies. While it is highly likely that only an extremely small fraction of these strategies would possess evolutionary robustness, the number of strategies requiring classification would still surpass our capacity. To enable an optimization algorithm to work in the domain of Boolean algebraic expression (strategies), we chose to represent players as multi-layer perceptrons (MLPs). Using non-linear activation functions such as the Rectified Linear Unit (ReLU), it has been widely demonstrated that MLPs with a single hidden layer are universal function approximators provided an adequate number of neurons [5]. Hence, MLPs with ReLU activations are also well-suited for representing boolean expressions. *Figure* 1 shows examples of logic gates that can be constructed using perceptrons, evidently the atoms of MLPs.

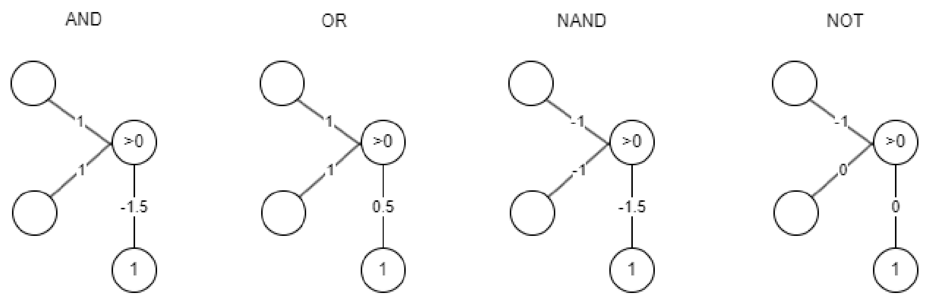


Figure 1: AND, OR, NAN, NOT gates constructed from perceptrons with a bias unit. *True*:=1, *False*:=-1.

[1]The number of opponent actions a player can retain in memory

While the NOT-gate does not need input from an additional bias unit, the other gates cannot be designed without it. Hence, our implementation uses bias units at the hidden and output layer respectively. The actions, i.e. responses and observations, were encoded by $\{-1 : C, 1 : D\}$. We initially used the encoding $\{0 : C, 1 : D\}$, but we could not make use of the MLPs' idle states this way. That is the players' responses to incomplete observations that are inherent to the beginning of the game. A player's action was generated by transforming the MLP's output using a threshold unit. A schematic of the MLP architecture that we used can be seen in *Figure* 2.

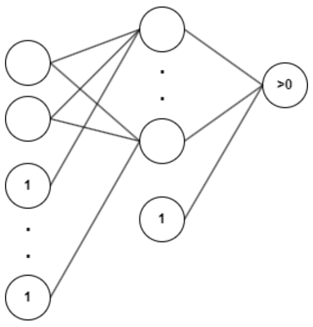


Figure 2: Schematic of the MLP architecture used to represent a player with a processing capacity of two. Hidden layer activations are ReLUs. There is a bias unit per hidden neuron and a bias unit for the output neuron. The output is mapped to -1,1 using a simple threshold unit. The total number of hidden neurons used is four for both a processing capacity of one and two.

The number of hidden neurons was determined empirically by sampling networks from a given architecture for a given processing capacity and verifying that the number of unique boolean expressions matches the theoretical maximum [2].

3.2 The Optimization Algorithm

We chose to use an evolutionary algorithm (EA) that acts on the fitness of a player measured by the accumulated reward during one generation. One generation comprises N players, playing against M other players (with replacement) K games of the PD. The parent selection is performed by sampling from an adaptable percentage of the fittest players considered the elite. Two parents generate one child. Both, the parents as well as their child transition to the next generation. We implemented mutation as well as a crossover. While mutation adds Gaussian noise with unit variance to the network weights (including the bias weights), crossover switches out entire logic gates, i.e. the incoming connections to hidden neurons (including the bias weights). We considered other forms of crossover such as switching the outgoing connections of input units or altering the second weight layer, yet, decided against those as they are less interpretable and more influential respectively. We consider our approach the golden middle among the named options. *Figure* 3 gives an overview of the options on how to implement crossover including the one used in our EA.

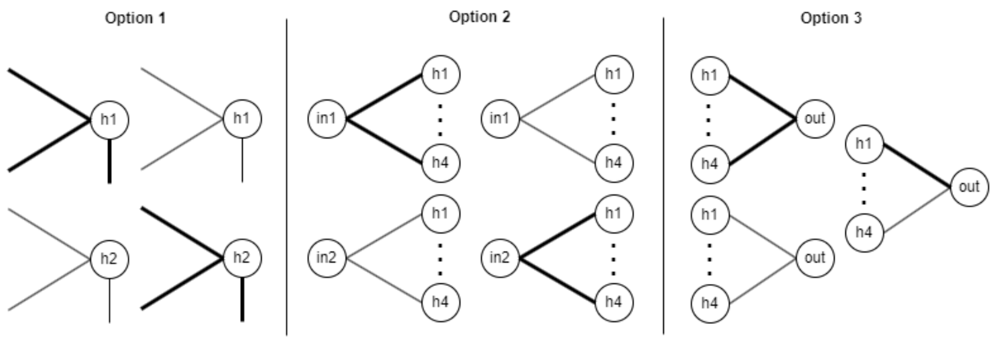


Figure 3: Possible implementations of crossover. Option 1 (our choice) exchanges logical gates. Option 2 exchanges contributions to logical gates. Option 3 alters the combination of logical gates.

We decided to exclude the processing capacity of a player from mutation because strategies available

[2]The theoretical maximum is given by: $2^{(2 \cdot 2^c - 1)}$, where $c :=$ processing capacity.

to players with lower processing capacities are inherently included in the strategies accessible to players with higher processing capacities.

3.3 The Detection Algorithm

Our detection algorithm acts as an additional player in each generation that cannot be selected away or altered through mutation. Depending on the processing capacity of players in a population the detector alters its strategy. The strategy is simply a chain of possible inputs that a player with a certain processing capacity can observe plus the player's idle state, represented by the input 0. *Figure* 4 exemplifies this procedure.

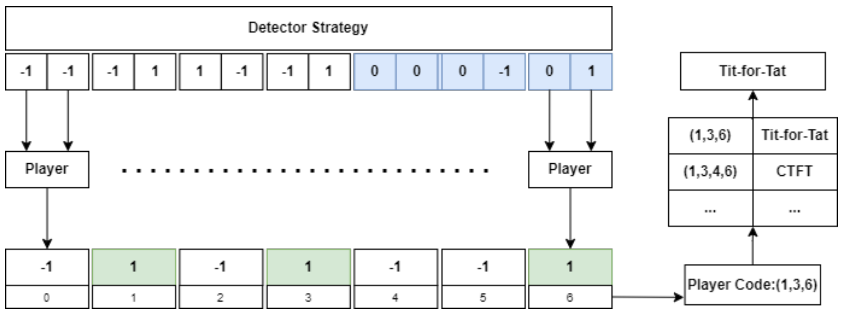


Figure 4: Schematic of the detection algorithm for a player with a processing capacity of two. The player is queried with every possible input (white) and its idle states (blue). The detector records the index of those inputs to whom the player responds with 1. From these indices a code is constructed and then compared to our database.

While the detection algorithm is rather simple, linking the player codes to strategies becomes difficult the more processing capacity a player has. Some strategies are only successful in certain edge cases. We made an effort to classify the ones that appeared frequently and neglected the ones that did not.

3.4 Data Collection and Simulation Setups

In order to evaluate the validity of our algorithm we executed several simulations with a fixed number of players, generation, matchups, games, and a set fitness function while varying the hyperparameters of the optimizer (mutation probability: μ, crossover probability: ρ, percentage of players belonging to the elite: η). We furthermore altered the processing capacity of players. To account for the stochasticity in the individual simulations, we ran each setting five times. We recorded the distribution dynamics of each run individually and their aggregates over five runs. We did the same for the reward dynamics. The table containing the specific simulation settings is shown below *table* 3.

Parameters	**Values**
players (N)	100
generations (L)	150
matchups (M)	60
games (K)	15
player fitness	$f(p_n) = \sum_{m=1}^{M} \sum_{k=1}^{K} r_{mk}$
c	1,2
μ	0.1, 0.5, 0.95
ρ	0, 0.3, 0.6
η	0.1, 0.5, 0.95

Table 3: Parameter settings used for simulations. Each simulation was run five times. 1350 simulations in total.

Given that the initial distribution of players is highly biased towards the strategies *Hawk* and *Dove* for both processing capacities *table* 4, we had to ensure that our initial strategy samples would be somewhat representative of the underlying strategy distribution. We could have also increased the number of runs per simulation but we chose to set the number of players to a reasonably high number instead. Based on the same reasoning we set the number of matchups for each player. The number of games is somewhat arbitrary. It is important to note, however, that it should not exceed the number of matchups as the dynamics will otherwise become significantly more deterministic due to the potential of over-representation of encounters with extremely rare strategies. We settled with a rather low number of games in an attempt to mitigate this risk.

Strategy	Percentage of players
Dove	34.451
TFT	7.471
CTFT	7.484
Hawk	34.204
RTFT	7.335
CRTFT	7.423
Undetermined	1.632

(a) Processing capacity=1

Strategy	Percentage of players
Dove	23.949
TFT	2.527
CTFT	2.504
Hawk	23.729
RTFT	2.523
GT	0.311
0,2,3,4,5,6	1.914
0,2,3	0.04
3	1.852
SG	1.266
CSG	0.104
Forgiving	1.339
2,3,6	0.575
0,2,3,4,5	1.335
0,2,3,5,6	0.088
1,6	1.353
Undetermined	34.591

(b) Processing capacity=2

Table 4: Strategy distribution of randomly initialized players

4 Results

In the initial segment of this section, we will examine the effectiveness of our methodology by assessing its capability of finding evolutionarily robust strategies. Subsequent subsections will delve deeper into its functionality by providing a detailed exploration of how our specific implementations of mutation, crossover, and elitism influence its performance.

4.1 Algorithmic Validity

Except for the rather extreme simulations that we ran, i.e. the ones that generated offspring from the top 10% of the population ($\epsilon = 0.1$), our algorithm was able to recover effective strategies for players with both processing capacities. At the same time, the found near Pareto-efficient Nash equilibria were usually not stable unless ϵ was 0.95. *Figure* 5 visualizes these findings.

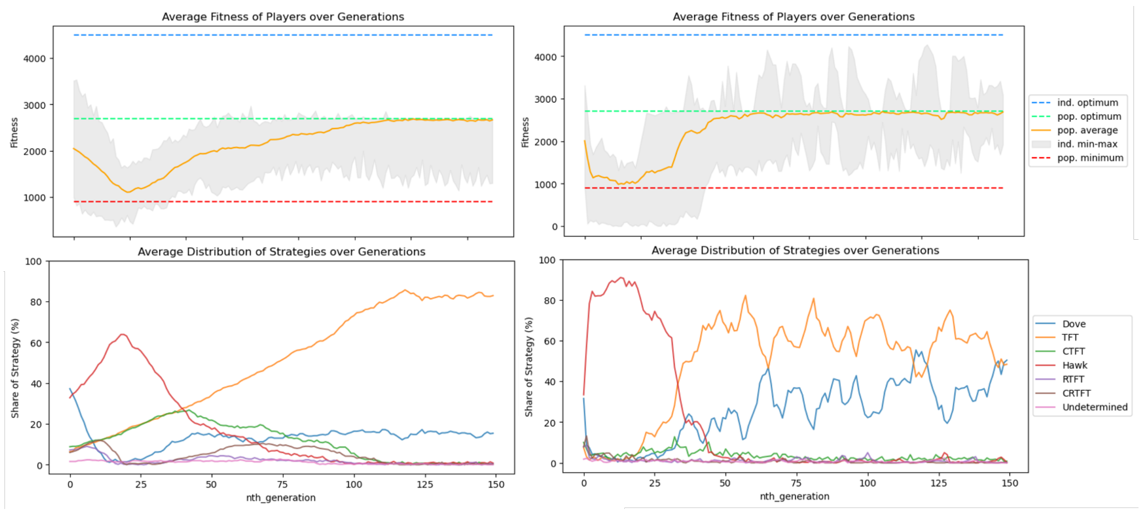


Figure 5: Dynamics of the average population rewards and strategy distributions for a population of players with a processing capacity of one. To the left, for the simulation setting ($\epsilon = 0.95, \mu = 1, \rho = 0.3$) and to the right for the setting ($\epsilon = 0.5, \mu = 1, \rho = 0.3$). Both settings indicate a decently fast recovery of effective strategies. Yet, the latter setting appears to yield less stable results.

We can clearly see that in both cases the population quickly converges to near-optimal Nash equilibria, denoted by the convergence of the population average to the population optimum together with low fluctuations in the corresponding dynamics of the average strategy distributions. The strategies found in these equilibria match the profile of strategies deemed effective in Axelrod's tournaments: Strategies that are conditionally cooperative, but at the same time capable of retaliation against defective strategies. This becomes more evident when looking at the dynamics of the average strategy distributions generated by the same setting for players with a processing capacity of two shown in *figure* 6

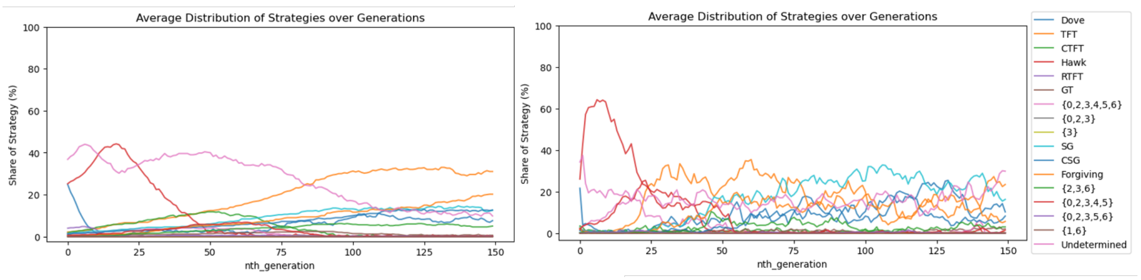


Figure 6: Dynamics of average strategy distributions for players with a processing capacity of two. Again, to the left, for the simulation setting ($\epsilon = 0.95, \mu = 1, \rho = 0.3$) and to the right for the setting ($\epsilon = 0.5, \mu = 1, \rho = 0.3$)

Besides *Tit-for-Tat* that was dominating the previously shown optimum, we can now observe the emergence of strategies that behave similarly. For, example, we can see *SG* and *CSG*, as well as *Forgiving* emerge. These are strategies that are cooperative at their core but implement retaliative mechanisms as well as mechanisms that enable them to escape deadlocks when defecting against *Tit-for-Tat*. Again, it becomes evident that our algorithm finds stable optima with $\epsilon = 0.95$ while being more stochastic with $\epsilon = 0.5$, showcasing the trade-off between exploration and exploitation.

4.2 Elitism

As discussed before, the proportion selected as the elite, ϵ, appears to have a strong effect on the stability of the found optima and the steadiness of convergence. When ϵ is low, i.e. few individuals are selected for breeding and survival, different strategies can quickly emerge but just as quickly disappear again. With high values for ϵ (0.5-0.95), simulations become significantly more predictable and less erratic. Figure 7 illustrates this effect. Both simulations have identical settings except for ϵ.

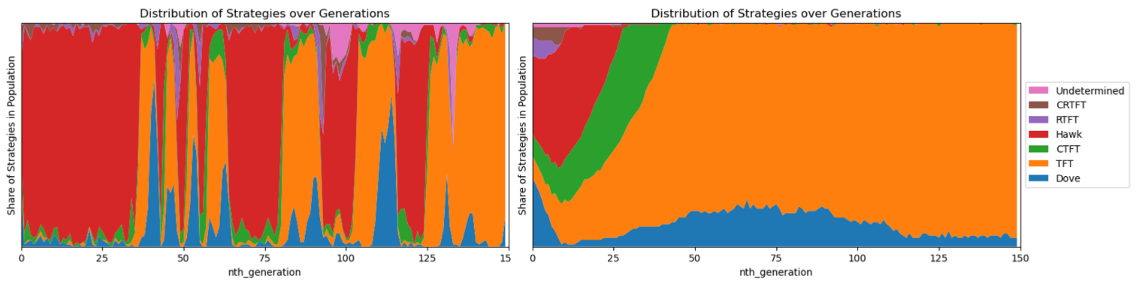


Figure 7: The influence of $\epsilon = 0.1$ (left) and $\epsilon = 0.95$ on the convergence behaviour of the evolutionary algorithm

4.3 Mutation & Crossover

Generally, the mutation rate, μ affects the probability that new strategies can be discovered and survive. We observed that the mutation rate by itself, i.e. in the absence of crossover ($\rho = 0$), has a significantly weaker effect than when combined with crossover as we can see in the figure below.

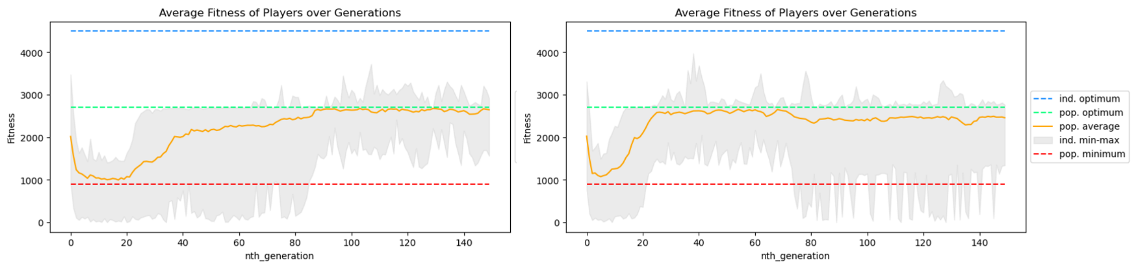


Figure 8: The influence of $\epsilon = 0.1$ (left) and $\epsilon = 0.95$ on the convergence behaviour of the evolutionary algorithm

Although our algorithm is able to find evolutionarily robust strategies around generation 20 until 80 (right plot), it happens to diverge again. In combination with crossover (left), the algorithm converges slower (also in other simulation settings) but remains in those locations of the parameter space that maps into effective strategies. A possible explanation for this could be that mutation steers optimization in parameter space while crossover does so in strategy space. Therefore, μ is dependent on the transition probabilities of strategies in parameter space while crossover is dependent on strategy similarity directly. We investigated the effect of μ on the transition probabilities between strategies in parameter space and found that, as expected, the higher μ the higher the entropy of the strategies' conditional transition distributions. This also explains the observed information loss and transition from a cooperative into a hostile game state when μ approaches one. Then, the odds of a hostile environment staying hostile are high. Players willing to cooperate can only thrive in an environment where other players willing to do the same. As such, it is required that multiple friendly players emerge in a single generation so that they can play off each other and outperform mutual defectors. When μ is low, the probability of this happening is also low. Section 7.3 shows the various mutation rates between strategies at different parameter settings. It becomes apparent that the dove and hawk strategies occupy a large part of the parameter space as all other strategies are likely to mutate into them, especially at higher mutation rates. The matrix containing the cosine similarities between strategies can be found in 7.4. We can see that in terms of cosine similarity *Hawk* and *Dove* are maximally different while other cooperative strategies with retaliative mechanisms exert similarity. When setting ρ to a value that allows the algorithm to exploit these similarities in strategy space rather than permute them we did observe robust and fast convergence behavior.

5 Conclusion

We demonstrated that an evolutionary algorithm that encodes players as MLPs and uses a strategy-preserving form of crossover can sustainably recover strategies that are robust and maximize welfare, i.e. effective strategies. Yet, we also revealed its sensitivity toward the tunable hyper-parameters μ, ρ, and η. Especially ϵ can be the deciding factor between robust and non-robust convergence. The hyperparameters μ and ρ, i.e. mutation rate and crossover probability represent complementary functions. While extreme values for both likely lead to erratic performance behavior, a well-balanced combination of both could potentially result in the discovery of highly effective new strategies given a less constrained design for players.

6 Discussion

There are many different choices to make when designing an agent in the IPD. In some of the related research, players in the IPD consider not only their opponent's recent moves but also their own, as well as both players' first moves. We found that certain strategies occupied bigger parts of the parameter space of the neural network, meaning that they can be more prevalent in the population despite not being optimal strategies. For this reason, future researchers could consider moving away from a feed-forward neural network altogether and experimenting with finite state machines, lookup tables, or even recurrent neural networks and transformers. The processing capacity of our network is low. It is very likely that better strategies can be found by taking more moves into consideration, but we faced the problem that the number of possible strategies increases exponentially with the processing capacity of the network, and it, therefore, becomes difficult to keep track of which strategies are actually performing well. Our implementation also does not account for probabilistic strategies. A player will always make the same move under the same circumstances. Certain viable strategies, such as the so-called Zero Determinant class of strategies, cannot emerge due to the deterministic nature of the neural network. This could be implemented by making our network output probabilities, rather than exact moves.

References

[1] Axelrod, R. (1980a). Effective choice in the prisoner's dilemma. *Journal of conflict resolution*, 24(1):3–25.

[2] Axelrod, R. (1980b). More effective choice in the prisoner's dilemma. *Journal of conflict resolution*, 24(3):379–403.

[3] Deutsch, M. (1958). Trust and suspicion. *Journal of conflict resolution*, 2(4):265–279.

[4] Harper, M., Knight, V., Jones, M., Koutsovoulos, G., Glynatsi, N. E., and Campbell, O. (2017). Reinforcement learning produces dominant strategies for the iterated prisoner's dilemma. *PloS one*, 12(12):e0188046.

[5] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

[6] Li, J., Hingston, P., and Kendall, G. (2011). Engineering design of strategies for winning iterated prisoner's dilemma competitions. *IEEE Transactions on Computational Intelligence and AI in Games*, 3:348–360.

[7] Rapoport, A. (1989). Prisoner's dilemma. *Game theory*, pages 199–204.

[8] Rapoport, A. (2012). *Game theory as a theory of conflict resolution*, volume 2. Springer Science & Business Media.

7 Appendix

7.1 Glossary of strategies

Strategy	Description
Hawk	D unconditionally
Dove	C unconditionally
Tit-For-Tat	C on first move, then copy opponents previous move
Cautious Tit-For-Tat	D on first move, then copy opponents previous move
Reverse Tit-For-Tat	C on first move, then opposite of opponents previous move
Cautious Reverse Tit-For-Tat	C on first move, then opposite of opponents previous move
Grim Trigger	C if opponent has never played D, otherwise D
Silent Grudge	C on first 2 moves, then copies opponents second to last move
Cautious Silent Grudge	D on first 2 moves, then copies opponents second to last move
Forgiving	C As long as opponent hasn't defected twice in last 2 moves
$x_1, x_2, ..., x_n$	Strategy can't easily be described but has unique identifier
Undetermined	Any other strategy

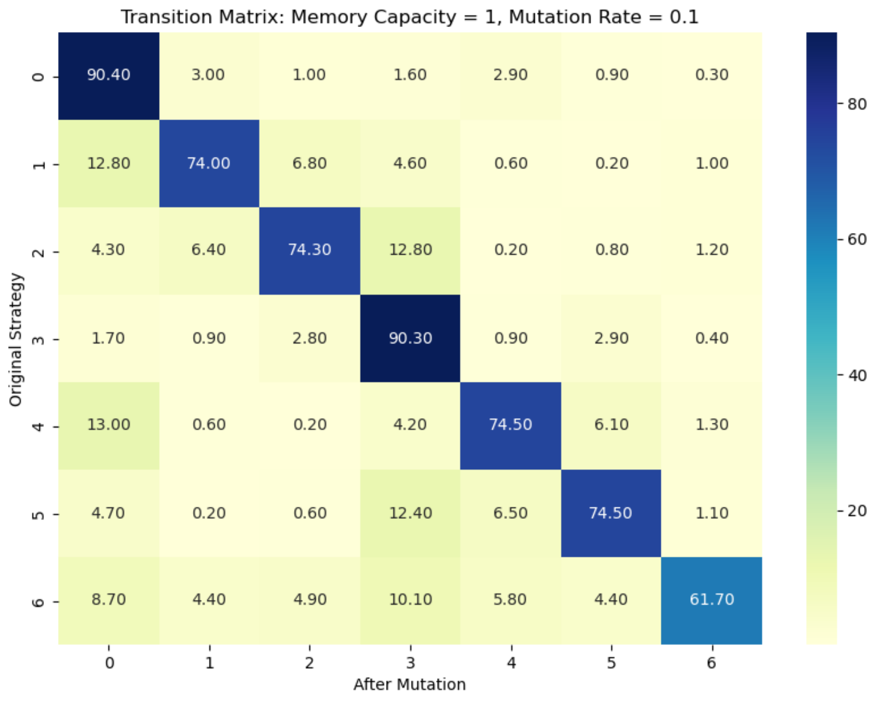
Table 5: Common strategies in the iterated PD

7.2 Index

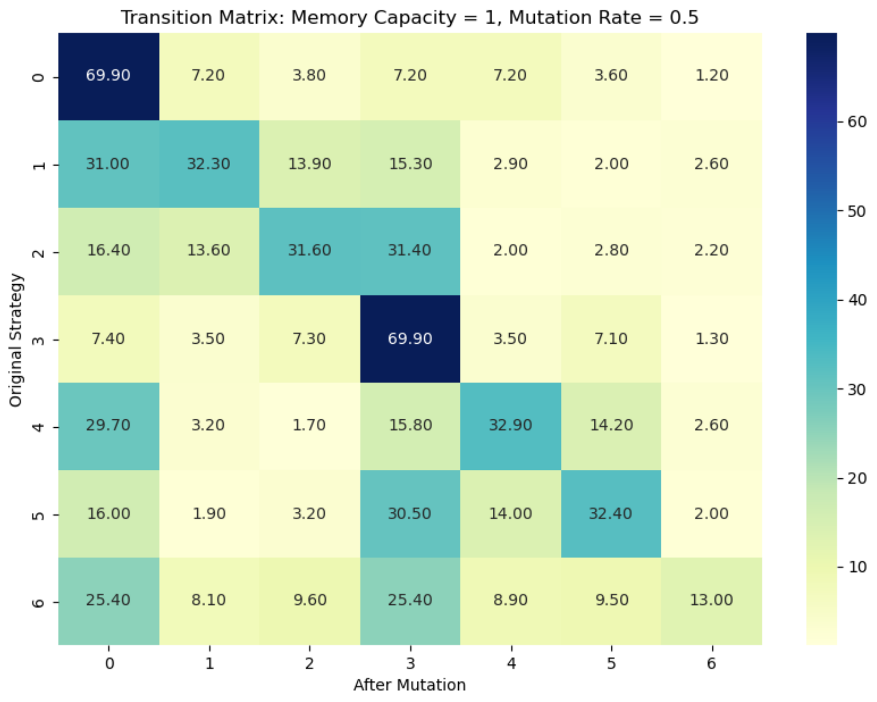
Strategy	**ID**
Dove	0
TFT	1
CTFT	2
Hawk	3
RTFT	4
GT	5
0,2,3,4,5,6	6
0,2,3	7
3	8
SG	9
CSG	10
Forgiving	11
2,3,6	12
0,2,3,4,5	13
0,2,3,5,6	14
1,6	15
Undetermined	16

Table 6: Strategy IDs

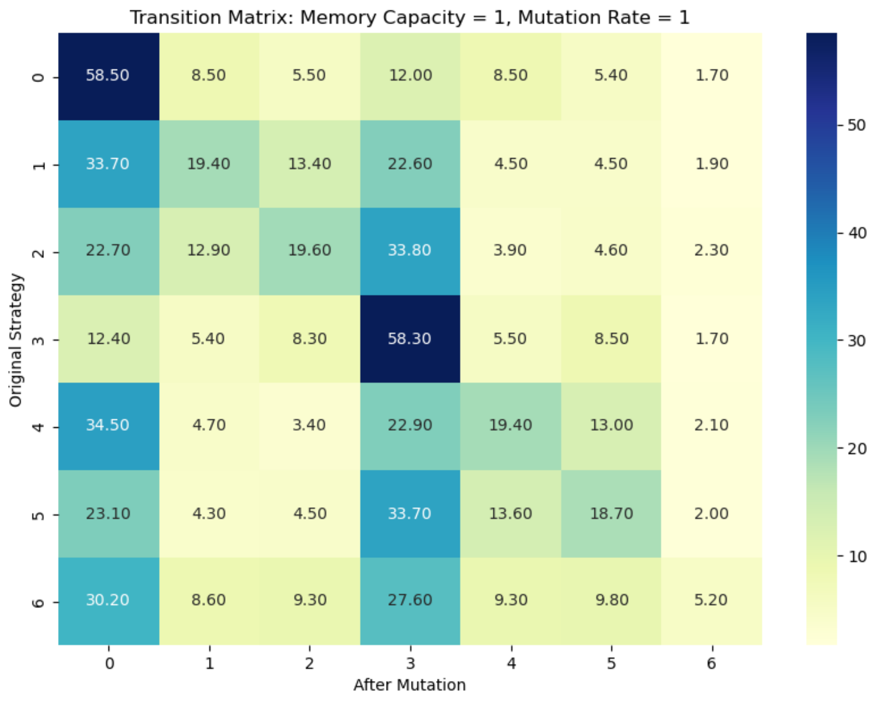
7.3 Mutation rates



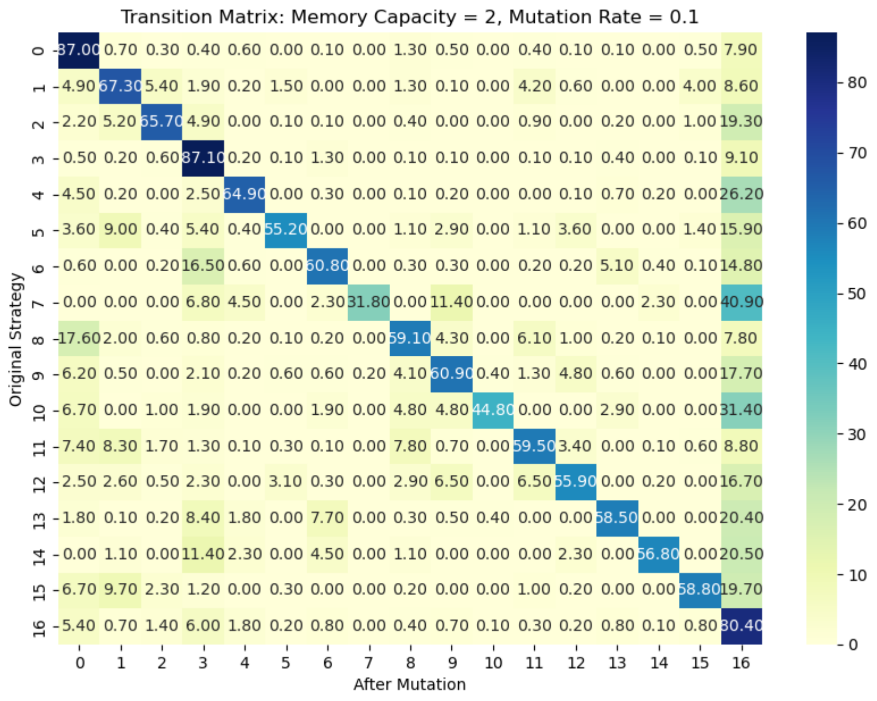
Original \ After	0	1	2	3	4	5	6
0	90.40	3.00	1.00	1.60	2.90	0.90	0.30
1	12.80	74.00	6.80	4.60	0.60	0.20	1.00
2	4.30	6.40	74.30	12.80	0.20	0.80	1.20
3	1.70	0.90	2.80	90.30	0.90	2.90	0.40
4	13.00	0.60	0.20	4.20	74.50	6.10	1.30
5	4.70	0.20	0.60	12.40	6.50	74.50	1.10
6	8.70	4.40	4.90	10.10	5.80	4.40	61.70



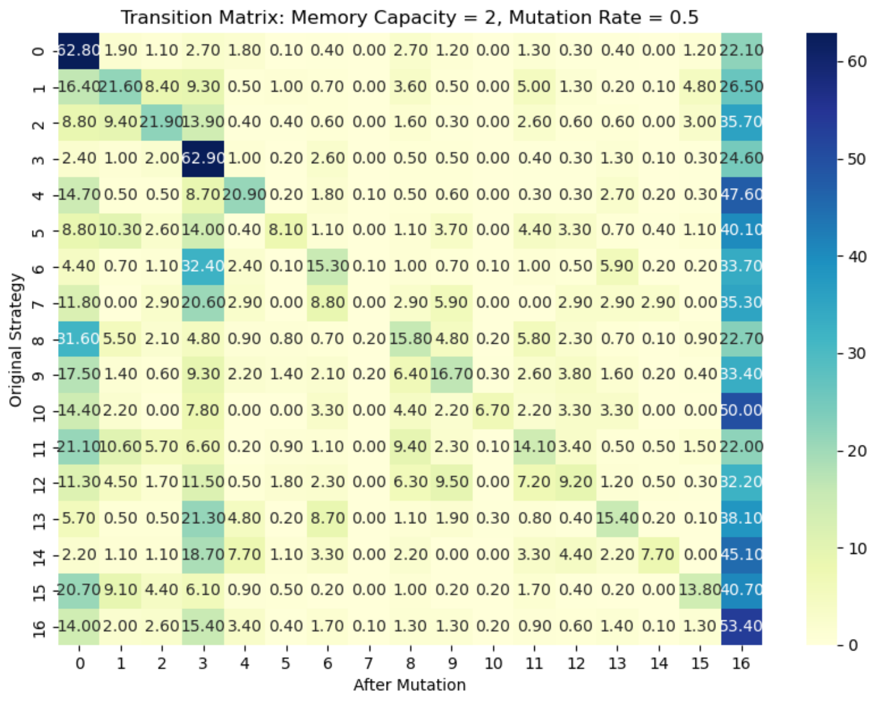
Original \ After	0	1	2	3	4	5	6
0	69.90	7.20	3.80	7.20	7.20	3.60	1.20
1	31.00	32.30	13.90	15.30	2.90	2.00	2.60
2	16.40	13.60	31.60	31.40	2.00	2.80	2.20
3	7.40	3.50	7.30	69.90	3.50	7.10	1.30
4	29.70	3.20	1.70	15.80	32.90	14.20	2.60
5	16.00	1.90	3.20	30.50	14.00	32.40	2.00
6	25.40	8.10	9.60	25.40	8.90	9.50	13.00



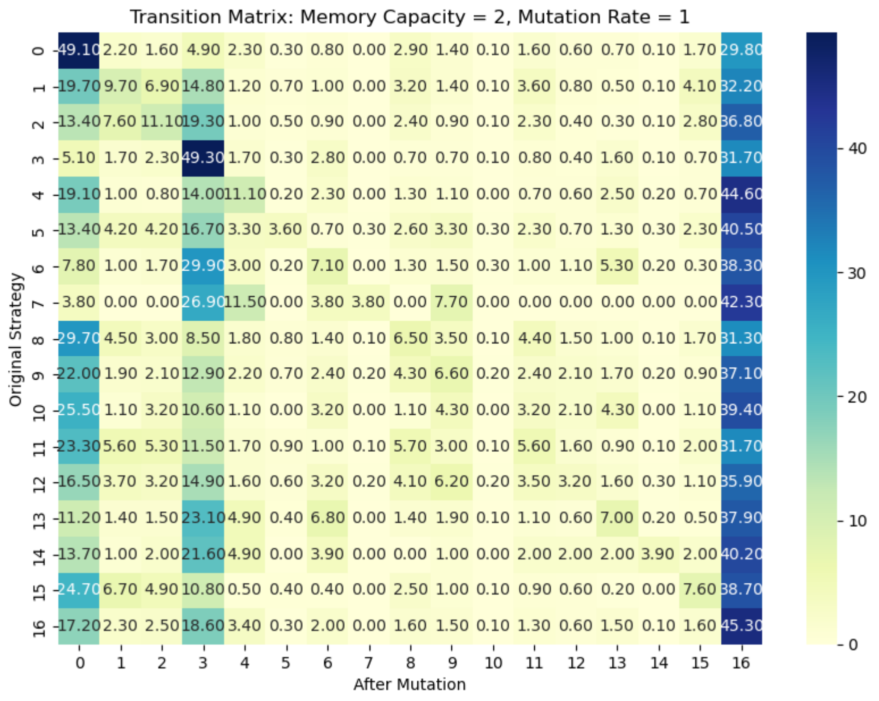
Transition Matrix: Memory Capacity = 1, Mutation Rate = 1
Original Strategy
After Mutation
0 1 2 3 4 5 6
0: 58.50 8.50 5.50 12.00 8.50 5.40 1.70
1: 33.70 19.40 13.40 22.60 4.50 4.50 1.90
2: 22.70 12.90 19.60 33.80 3.90 4.60 2.30
3: 12.40 5.40 8.30 58.30 5.50 8.50 1.70
4: 34.50 4.70 3.40 22.90 19.40 13.00 2.10
5: 23.10 4.30 4.50 33.70 13.60 18.70 2.00
6: 30.20 8.60 9.30 27.60 9.30 9.80 5.20
50 40 30 20 10



Transition Matrix: Memory Capacity = 2, Mutation Rate = 0.1
Original Strategy
After Mutation
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0: 87.00 0.70 0.30 0.40 0.60 0.00 0.10 0.00 1.30 0.50 0.00 0.40 0.10 0.10 0.00 0.50 7.90
1: 4.90 67.30 5.40 1.90 0.20 1.50 0.00 0.00 1.30 0.10 0.00 4.20 0.60 0.00 0.00 4.00 8.60
2: 2.20 5.20 65.70 4.90 0.00 0.10 0.10 0.00 0.40 0.00 0.00 0.90 0.00 0.20 0.00 1.00 19.30
3: 0.50 0.20 0.60 87.10 0.20 0.10 1.30 0.00 0.10 0.10 0.00 0.10 0.10 0.40 0.00 0.10 9.10
4: 4.50 0.20 0.00 2.50 64.90 0.00 0.30 0.00 0.10 0.20 0.00 0.00 0.10 0.70 0.20 0.00 26.20
5: 3.60 9.00 0.40 5.40 0.40 55.20 0.00 0.00 1.10 2.90 0.00 1.10 3.60 0.00 0.00 1.40 15.90
6: 0.60 0.00 0.20 16.50 0.60 0.00 60.80 0.00 0.30 0.30 0.00 0.20 0.20 5.10 0.40 0.10 14.80
7: 0.00 0.00 0.00 6.80 4.50 0.00 2.30 31.80 0.00 11.40 0.00 0.00 0.00 0.00 2.30 0.00 40.90
8: 17.60 2.00 0.60 0.80 0.20 0.10 0.20 0.00 59.10 4.30 0.00 6.10 1.00 0.20 0.10 0.00 7.80
9: 6.20 0.50 0.00 2.10 0.20 0.60 0.60 0.20 4.10 60.90 0.40 1.30 4.80 0.60 0.00 0.00 17.70
10: 6.70 0.00 1.00 1.90 0.00 0.00 1.90 0.00 4.80 4.80 44.80 0.00 0.00 2.90 0.00 0.00 31.40
11: 7.40 8.30 1.70 1.30 0.10 0.30 0.10 0.00 7.80 0.70 0.00 59.50 3.40 0.00 0.10 0.60 8.80
12: 2.50 2.60 0.50 2.30 0.00 3.10 0.30 0.00 2.90 6.50 0.00 6.50 55.90 0.00 0.20 0.00 16.70
13: 1.80 0.10 0.20 8.40 1.80 0.00 7.70 0.00 0.30 0.50 0.40 0.00 0.00 58.50 0.00 0.00 20.40
14: 0.00 1.10 0.00 11.40 2.30 0.00 4.50 0.00 1.10 0.00 0.00 0.00 2.30 0.00 56.80 0.00 20.50
15: 6.70 9.70 2.30 1.20 0.00 0.30 0.00 0.00 0.20 0.00 0.00 1.00 0.20 0.00 0.00 58.80 19.70
16: 5.40 0.70 1.40 6.00 1.80 0.20 0.80 0.00 0.40 0.70 0.10 0.30 0.20 0.80 0.10 0.80 80.40
80 70 60 50 40 30 20 10 0

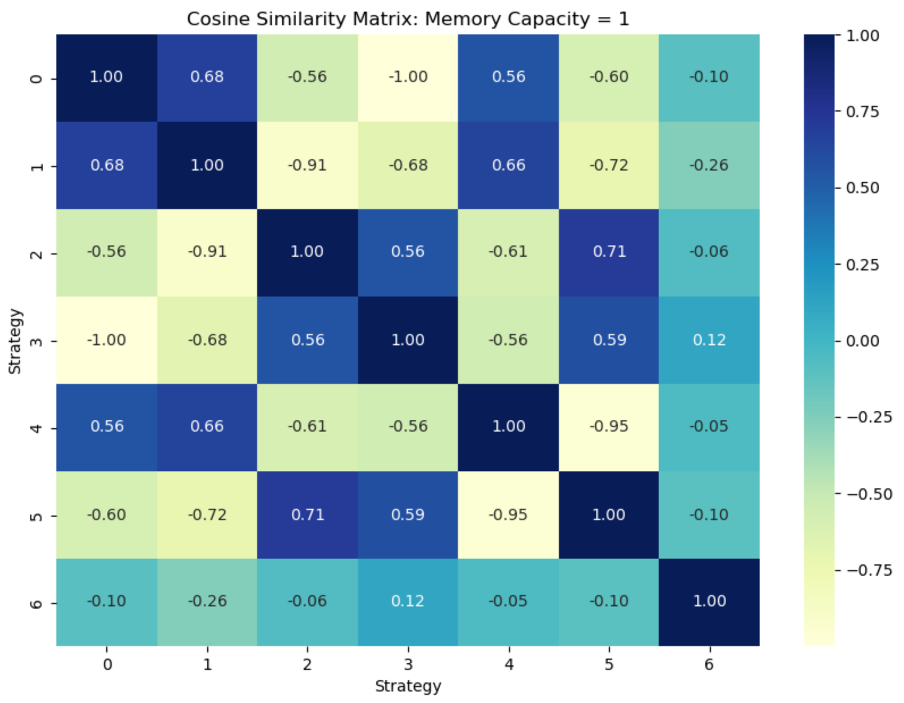


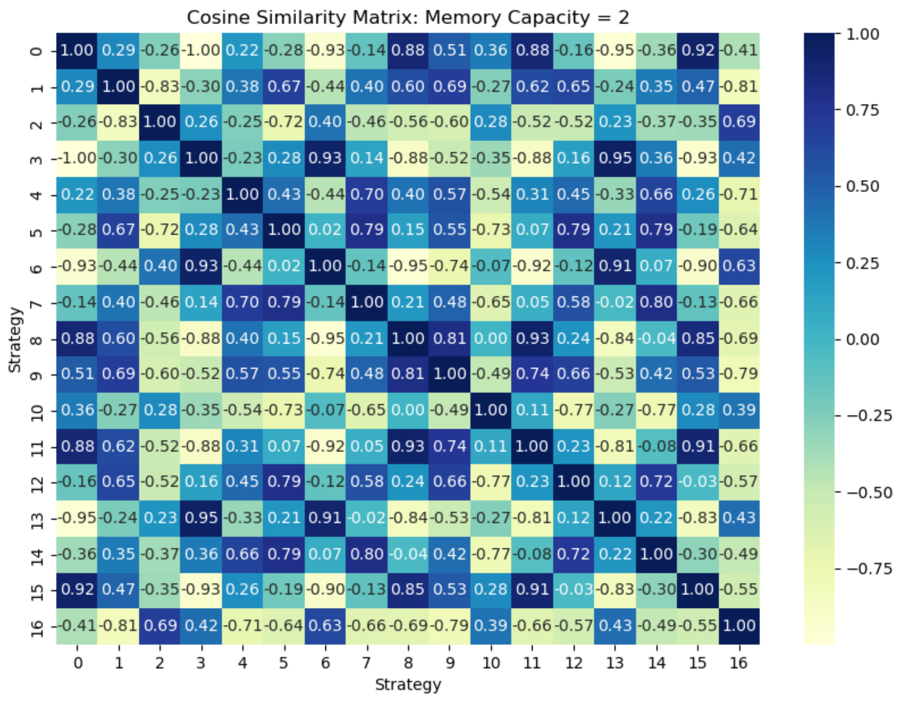
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	62.80	1.90	1.10	2.70	1.80	0.10	0.40	0.00	2.70	1.20	0.00	1.30	0.30	0.40	0.00	1.20	22.10
1	16.40	21.60	8.40	9.30	0.50	1.00	0.70	0.00	3.60	0.50	0.00	5.00	1.30	0.20	0.10	4.80	26.50
2	8.80	9.40	21.90	13.90	0.40	0.40	0.60	0.00	1.60	0.30	0.00	2.60	0.60	0.60	0.00	3.00	35.70
3	2.40	1.00	2.00	62.90	1.00	0.20	2.60	0.00	0.50	0.50	0.00	0.40	0.30	1.30	0.10	0.30	24.60
4	14.70	0.50	0.50	8.70	20.90	0.20	1.80	0.10	0.50	0.60	0.00	0.30	0.30	2.70	0.20	0.30	47.60
5	8.80	10.30	2.60	14.00	0.40	8.10	1.10	0.00	1.10	3.70	0.00	4.40	3.30	0.70	0.40	1.10	40.10
6	4.40	0.70	1.10	32.40	2.40	0.10	15.30	0.10	1.00	0.70	0.10	1.00	0.50	5.90	0.20	0.20	33.70
7	11.80	0.00	2.90	20.60	2.90	0.00	8.80	0.00	2.90	5.90	0.00	0.00	2.90	2.90	2.90	0.00	35.30
8	31.60	5.50	2.10	4.80	0.90	0.80	0.70	0.20	15.80	4.80	0.20	5.80	2.30	0.70	0.10	0.90	22.70
9	17.50	1.40	0.60	9.30	2.20	1.40	2.10	0.20	6.40	16.70	0.30	2.60	3.80	1.60	0.20	0.40	33.40
10	14.40	2.20	0.00	7.80	0.00	0.00	3.30	0.00	4.40	2.20	6.70	2.20	3.30	3.30	0.00	0.00	50.00
11	21.10	10.60	5.70	6.60	0.20	0.90	1.10	0.00	9.40	2.30	0.10	14.10	3.40	0.50	0.50	1.50	22.00
12	11.30	4.50	1.70	11.50	0.50	1.80	2.30	0.00	6.30	9.50	0.00	7.20	9.20	1.20	0.50	0.30	32.20
13	5.70	0.50	0.50	21.30	4.80	0.20	8.70	0.00	1.10	1.90	0.30	0.80	0.40	15.40	0.20	0.10	38.10
14	2.20	1.10	1.10	18.70	7.70	1.10	3.30	0.00	2.20	0.00	0.00	3.30	4.40	2.20	7.70	0.00	45.10
15	20.70	9.10	4.40	6.10	0.90	0.50	0.20	0.00	1.00	0.20	0.20	1.70	0.40	0.20	0.00	13.80	40.70
16	14.00	2.00	2.60	15.40	3.40	0.40	1.70	0.10	1.30	1.30	0.20	0.90	0.60	1.40	0.10	1.30	53.40



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	49.10	2.20	1.60	4.90	2.30	0.30	0.80	0.00	2.90	1.40	0.10	1.60	0.60	0.70	0.10	1.70	29.80
1	19.70	9.70	6.90	14.80	1.20	0.70	1.00	0.00	3.20	1.40	0.10	3.60	0.80	0.50	0.10	4.10	32.20
2	13.40	7.60	11.10	19.30	1.00	0.50	0.90	0.00	2.40	0.90	0.10	2.30	0.40	0.30	0.10	2.80	36.80
3	5.10	1.70	2.30	49.30	1.70	0.30	2.80	0.00	0.70	0.70	0.10	0.80	0.40	1.60	0.10	0.70	31.70
4	19.10	1.00	0.80	14.00	11.10	0.20	2.30	0.00	1.30	1.10	0.00	0.70	0.60	2.50	0.20	0.70	44.60
5	13.40	4.20	4.20	16.70	3.30	3.60	0.70	0.30	2.60	3.30	0.30	2.30	0.70	1.30	0.30	2.30	40.50
6	7.80	1.00	1.70	29.90	3.00	0.20	7.10	0.00	1.30	1.50	0.30	1.00	1.10	5.30	0.20	0.30	38.30
7	3.80	0.00	0.00	26.90	11.50	0.00	3.80	3.80	0.00	7.70	0.00	0.00	0.00	0.00	0.00	0.00	42.30
8	29.70	4.50	3.00	8.50	1.80	0.80	1.40	0.10	6.50	3.50	0.10	4.40	1.50	1.00	0.10	1.70	31.30
9	22.00	1.90	2.10	12.90	2.20	0.70	2.40	0.20	4.30	6.60	0.20	2.40	2.10	1.70	0.20	0.90	37.10
10	25.50	1.10	3.20	10.60	1.10	0.00	3.20	0.00	1.10	4.30	0.00	3.20	2.10	4.30	0.00	1.10	39.40
11	23.30	5.60	5.30	11.50	1.70	0.90	1.00	0.10	5.70	3.00	0.10	5.60	1.60	0.90	0.10	2.00	31.70
12	16.50	3.70	3.20	14.90	1.60	0.60	3.20	0.20	4.10	6.20	0.20	3.50	3.20	1.60	0.30	1.10	35.90
13	11.20	1.40	1.50	23.10	4.90	0.40	6.80	0.00	1.40	1.90	0.10	1.10	0.60	7.00	0.20	0.50	37.90
14	13.70	1.00	2.00	21.60	4.90	0.00	3.90	0.00	0.00	1.00	0.00	2.00	2.00	2.00	3.90	2.00	40.20
15	24.70	6.70	4.90	10.80	0.50	0.40	0.40	0.00	2.50	1.00	0.10	0.90	0.60	0.20	0.00	7.60	38.70
16	17.20	2.30	2.50	18.60	3.40	0.30	2.00	0.00	1.60	1.50	0.10	1.30	0.60	1.50	0.10	1.60	45.30

7.4 Cosine Similarity


Cosine Similarity Matrix: Memory Capacity = 1
Strategy
Strategy


Cosine Similarity Matrix: Memory Capacity = 2
Strategy
Strategy