UTRECHT UNIVERSITY

Department of Mathematics

Bachelor Thesis Mathematics and Applications

Dominant Strategies in a Memory-Two Repeated Prisoner's Dilemma

Supervisor:

Prof. dr. ir. J.E. Frank

Author:

Senne Versteeg

June 2024

Abstract

This thesis explores dominant strategies in a memory-two iterated Prisoner's dilemma (IPD) using Markov chains and numerical simulations. By focusing on the memory-2 IPD, where each player remembers the outcomes of the previous two games, the research aims to identify optimal strategies. The findings demonstrate how increased memory affects strategic interactions and highlight common characteristics shared by dominant strategies. This work extends the understanding of strategic behavior in IPD, building upon prior studies on memory-one IPD.

Contents

1 Introduction

Created by M. Flood and M. Dreshner in the '50s, the Prisoner's dilemma is the leading example for a 2-player strategic game, and for analyzing the evolution of cooperative and defective behaviour. The two players of the Prisoner's dilemma have to choose between a strategy of cooperation (C) or defection (D), where payoffs of the players depend on the choices of both agents. Specifically, mutual cooperation yields a reward of R for each player, while mutual defection results in a payoff of P. If one player defects while the other cooperates, the defector gains T, and the cooperator receives S, where the inequality $T > R > P > S$ holds.

In the non-iterated version of the Prisoner's dilemma, defection is known to be the best strategy. However, this conclusion changes in the iterated version of the Prisoner's dilemma (IPD). In the iterated scenario, the same two agents repeatedly play the Prisoner's dilemma, adjusting their strategies based on the payoffs received and the actions taken in previous rounds. This iterative process allows the agents to learn from each other's behavior and adapt their strategies accordingly.

When players can recall the outcomes of the previous n games, the IPD is referred to as a memory-n game. This thesis demonstrates that the memory-n IPD can be modeled as a Markov chain. We will focus specifically on the memory-2 IPD, where each player remembers the outcomes of the preceding 2 games. Utilizing the Markovian properties of the memory-2 IPD, we aim to calculate the average long-term payoff for each player through numerical simulations.

The purpose of this paper is to find the best strategies for the memory-2 IPD, by using Markov chains and numerical simulations. We discuss these results, and compare them to the memory-1 IPD, which has already been studied in numerous works, such as *"Strategies in the stochastic iterated prisoner's dilemma"* by S. Li, *"A Strategy of win-stay, lose-shift that outperforms tit-for-tat in the Prisoner's dilemma"* by M. Nowak and K. Sigmund [1] and *'Effects of increasing the number of players and memory size in the iterated Prisoner's dilemma: a numerical approach'* by CH. Hauert and H. G. Schuster [4].

In Section 1, we introduce the Prisoner's dilemma, its significance in game theory, and the transition from non-iterated to iterated versions. It sets the stage for the focus on memory-2 IPD and outlines the purpose and scope of the research.

Section 2 provides a detailed description of the formal setup of the IPD. It explains the theoretical underpinnings of the game, including payoff matrices and strategy formulations. This section introduces the concept of Markov chains and how they are utilized to model the memory-2 IPD. It discusses the state space and transition probabilities in the context of memory-2 strategies. Additionally, this section outlines the simulation approach, including the design and implementation of the numerical simulations used to explore the strategic interactions in memory-2 IPD.

Section 3 presents the outcomes of the numerical simulations, identifying the dominant strategies in the memory-2 IPD. It includes a comparison of these results with those from memory-1 IPD studies.

In Section 4, the main findings of the thesis are summarized and interpreted, emphasizing the impact of increased memory on strategic interactions. It discusses the shared characteristics of dominant strategies and suggests directions for future research, highlighting that further advancements are limited by computational power.

2 Methodology and framework

2.1 Non-iterated Prisoner's dilemma

Firstly, we describe the non-iterated Prisoner's dilemma formally, to make the reader familiar with this game. We will show what is the best strategy in the non-iterated Prisoner's dilemma as well. Recall from the introduction that both players can choose between a strategy of cooperation (C) or a defective strategy (D). We keep track of the chosen actions and the corresponding payoffs of both players in tuples (a_1, a_2) and (p_1, p_2) respectively, where subscript 1 refers to the first agent and subscript 2 to the second. For example, a chosen strategy of $(a_1, a_2) = (C, C)$ will give $(p_1, p_2) = (R, R)$ as a payoff. Furthermore, we require[1] that $2R > T + S$. In table form:

	$a_2 = C$	$a_2 = D$
$a_1 = C$	$(p_1, p_2) = (R, R)$	$(p_1, p_2) = (S, T)$
$a_1 = D$	$(p_1, p_2) = (T, S)$	$(p_1, p_2) = (P, P)$

In this section, we introduce the notion of *payoff function* and *Nash equilibrium*, to provide a method of comparing payoffs and strategies.

Definition 2.1 (Payoff function for two players). *Let A_i be the set of actions player i can choose from. The payoff function for player i is a function $u_i(a_1, a_2) : A_1 \times A_2 \to \mathbb{R}$, representing the payoff for player i considering the actions of both players.*

For the Prisoner's dilemma, we have that $A_1 = A_2 = \{C, D\}$. For example, $u_1(C, D) = S$ and $u_2(C, D) = T$. To determine what is the best strategy in a game, we introduce the definition of the *Nash equilibrium*:

[1] The payoff for two agents both cooperating in both rounds is $2R$. The payoff of one defecting agent exploiting the other in the first round and vice versa in the other round is $T + S$. Therefore, to ensure mutual cooperation is optimal, we require $2R > T + S$.

Definition 2.2 (**Nash equilibrium**). *Let $A = A_1 \times \cdots \times A_N$ be the cartesian product of the sets of actions of all N players. We use the notation a^*_{-i} to denote the $N-1$ strategies of all players except i. Let $a^* = (a^*_i, a^*_{-i}) \in A$ be a strategy profile, consisting of 1 strategy for each player. The strategy profile $a^* \in A$ is a Nash equilibrium if*

$$u_i(a^*_i, a^*_{-i}) \geq u_i(a_i, a^*_{-i}), \quad \forall a_i \in A_i.$$

A Nash equilibrium is a strict Nash equilibrium if the inequality is strict, i.e. there is a unique best strategy.

Informally, a strategy profile is a Nash equilibrium if no player can gain by unilaterally changing their own strategy. With this definition, we can see that $a^* = (D, D)$ is a strict Nash equilibrium: no player can gain by changing their strategy to C. Therefore, the best option is to always defect from the opponent, and playing D guarantees the highest payoff, regardless of what the other player chooses.

However, the IPD fundamentally changes the strategic landscape compared to the non-iterated Prisoner's Dilemma. In the one-shot game, defecting is a dominant strategy for both players because it maximizes their immediate payoff regardless of the opponent's choice. The iterated version of the game allows for the development of trust and reciprocity, leading to higher cumulative payoffs from cooperation compared to the short-term gains from defection.

2.2 Markov chain

To facilitate the analysis of the IPD, it is useful to formulate the problem we are dealing with mathematically. We will formulate the IPD using Markov chain and its properties. As briefly remarked in the Introduction, the memory-1 IPD is a Markov chain. The memory-2 IPD can be formulated as a Markov chain as well. To show this, we introduce the definition of a Markov chain. We will also define the transition matrix of a Markov chain, and show how to describe the IPD as a Markov chain.

Definition 2.3 (**State space, Markov chain**). *A Markov chain is a stochastic process* $\{V_n\}_{n\geq 0}$. *The possible values* V_i *can attain form a countable set* S *called the state space. Furthermore, the Markov property is satisfied: If* V_i *is the state the system attains in time step i:*

$$\mathbb{P}(V_{i+1} = v_{i+1} | V_i = v_i, V_{i-1} = v_{i-1}, \ldots, V_1 = v_1) = \mathbb{P}(V_{i+1} = v_{i+1} | V_i = v_i).$$

Often, the last attribute is informally stated as, "The future given the present is not dependent on the past". Given a discrete Markov chain, we can introduce the definition of transition matrix:

Definition 2.4 (**one-step transition matrix**). *A transition matrix (or stochastic matrix) is a square matrix representing the probabilities of changing from one state to the other in a Markov chain. Specifically, if the state space of a Markov chain is* $S = \{s_1, s_2, \ldots, s_d\}$, *and we define* $p_{ij} = \mathbb{P}(V_{n+1} = s_j | V_n = s_i)$, *then the one-step transition matrix* $P \in [0,1]^{d\times d}$ *is given by*

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1d} \\ p_{21} & p_{22} & \cdots & p_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ p_{d1} & p_{d2} & \cdots & p_{dd} \end{bmatrix}.$$

For simplicity, we sometimes write $\mathbb{P}(V_{n+1} = s_j | V_n = s_i) = \mathbb{P}(s_i \to s_j)$.

It is also trivial to observe the following:

Corollary 2.5. *The sum of any row,* $\sum_{j=1}^{d} p_{ij}$, *represents the probability of transi-*

tioning from state s_i to any other state including s_i itself. Therefore, the sum of the rows of a transition matrix must be equal to one:

$$\sum_{j=1}^{d} p_{ij} = \sum_{j=1}^{d} \mathbb{P}(s_j|s_i) = 1. \tag{2.2.1}$$

The probability of the system being in a certain state can be completely determined by the intial state and the one-step transition matrix: if the initial state is a vector $\boldsymbol{x}_0$, the next state is given by

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 P.$$

We can calculate any n-step leap in a Markov chain using a transition matrix as well, by left multiplying the state vector $\boldsymbol{x}_0$ with the transition matrix n times. If we want to find the state vector $\boldsymbol{x}_n$, it can be calculated by

$$\boldsymbol{x}_n = \boldsymbol{x}_0 P^n.$$

2.3 IPD as a Markov chain

Given these definitions, we can begin to construct the IPD as a Markov chain. To make the reader familiar with the memory-2 case of the IPD, we first start with a simpler variant, the memory-1 IPD.

2.3.1 Memory-1 IPD

We will need to keep track of the played games in the IPD. We denote the played games by a sequence $\{U_n\}_{n\geq 1}$. We formulate every outcome of a game by a two-letter sequence $\{CC, CD, DC, DD\}$, indicating the actions of the two players. The first letter represents the action of player 1, and the second letter that of player 2. Since $U_i \in \{CC, CD, DC, DD\}$, we have

$$\begin{aligned} S &= \{s_1, s_2, s_3, s_4\} \\ &= \{CC, CD, DC, DD\}. \end{aligned}$$

U_1 is the outcome of the first game, U_2 that of the second, etc. These 4 elements are all the possible outcomes of 1 game. Since the outcome of the next game is only dependent on the previous outcome, the sequence $\{U_n\}_{n\geq 1}$ is a Markov chain.
To contruct the transition matrix associated with the memory-1 IPD, we define a vector $\boldsymbol{p}$ for player 1.

$$\boldsymbol{p} = (p_1, p_2, p_3, p_4), \qquad p_i \in [0, 1].$$

The elements of $\boldsymbol{p}$ correspond to the probability of playing C after observing an outcome in $\{CC, CD, DC, DD\}$; p_i is the probability of cooperating after a previous outcome of $s_i \in S$. We denote the played games from the perspective of player 1. Therefore, the vector $\boldsymbol{q}$ of player 2 will look slightly different:

$$\boldsymbol{q} = (q_1, q_3, q_2, q_4), \qquad q_i \in [0, 1].$$

For example, if the previous game played had an outcome of CD, player 1 will cooperate again with probablility p_2. Since player 2 views the game in their own perspective, they observed an outcome of DC. Hence, they will

cooperate again with a probablility of q_3. The transition matrix P of this game is now completely determined by the strategy vectors $\boldsymbol{p}$ and $\boldsymbol{q}$. Since there are 4 possible states for the system to be in, the transition matrix P will be a 4×4 matrix:

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{34} & p_{44} \end{bmatrix}.$$

Each element p_{ij} represents the probability of a transition from state s_i to state s_j. For instance, $\mathbb{P}(DC \to DD) = p_{34} = (1 - p_3)(1 - q_2)$. Player 1 observed DC and plays D with probability $(1 - p_3)$. From their perspective, player 2 observed a previous game of CD. They will play D with a probability of $(1 - q_2)$. By considering every transition like this, P becomes:

$$\begin{bmatrix} p_1q_1 & p_1(1-q_1) & (1-p_1)q_1 & (1-p_1)(1-q_1) \\ p_2q_3 & p_2(1-q_3) & (1-p_2)q_3 & (1-p_2)(1-q_3) \\ p_3q_2 & p_3(1-q_2) & (1-p_3)q_2 & (1-p_3)(1-q_2) \\ p_4q_4 & p_4(1-q_4) & (1-p_4)q_4 & (1-p_4)(1-q_4) \end{bmatrix}.$$

2.3.2 Memory-2 IPD

The contruction of the memory-2 IPD shares similarities with the memory-1 IPD. If we denote the sequence of played games by $\{U_n\}_{n\geq 1}$, as in the memory-1 case, the memory-2 variant of the IPD is initially not a Markov chain; the state U_{n+1} is not only dependent on the previous outcome U_n, but also on the second-to-last outcome U_{n-1}.

The key insight to make the system Markovian is to consider the outcome of the previous round *and* the outcome of the current round as a new process, $\{V_n\}_{n\geq 1}$. Since there are 4 possible outcomes of playing one game, there are $4^2 = 16$ ways to play two games, and the state space of $\{V_n\}$ contains 16 elements:

$$\begin{aligned} S &= \{s_1, s_2, \ldots, s_{16}\} \\ &= \{CCCC, CCCD, CCDC, CCDD, CDCC, CDCD, CDDC, CDDD, \\ &\quad DCCC, DCCD, DCDC, DCDD, DDCC, DDCD, DDDC, DDDD\}. \end{aligned}$$

We use the notation $e_1e_2e_3e_4$ with $e_i \in \{C, D\}$ to represent a state where the previous outcome was e_1e_2 and the current outcome is e_3e_4. This way, since any element of the state space contains all the information of the previous 2 played rounds, the process $\{U_n\}_{n\geq 1}$ has become Markovian by converting it to the new process $\{V_n\}_{n\geq 1}$.

The next step is to construct a transition matrix P for the memory-2 IPD. To do this, we want to keep track of the strategy of every player. We do this by using a row vector $\boldsymbol{p}$ for player 1 and a row vector $\boldsymbol{q}$ for player 2, similar to the memory-1 case. Since there are now 16 possible states to be considered, $\boldsymbol{p}$ and $\boldsymbol{q}$ also contain 16 elements.

$$\boldsymbol{p} = (p_1, p_2, \ldots, p_{15}, p_{16}), \qquad p_i \in [0, 1].$$

Similar to the memory-1 case, an outcome of $e_1e_2e_3e_4$ from the perspective of player 1 will be an outcome of $e_2e_1e_4e_3$ from the perspective of player 2. (For example, $DDDC$ becomes $DDCD$ in player 2's perspective.) Therefore,

the row vector $\boldsymbol{q}$ will look slightly different:

$$\boldsymbol{q} = (q_1, q_3, q_2, q_4, q_9, q_{11}, q_{10}, q_{12}, q_5, q_7, q_6, q_8, q_{13}, q_{15}, q_{14}, q_{16}), \qquad q_i \in [0,1].$$

With this adjustment, the ith element of $\boldsymbol{p}$ and the ith element of $\boldsymbol{q}$ are the probabilities of cooperating after observing an outcome of $s_i \in S$, from the perspective of player 1.

Every state in the process represents the outcome of the current and the previous game, so there is an overlap of 1 played game between the states. Therefore, our transition matrix P becomes sparse:

$$\mathbb{P}(V_{n+1} = e_1e_2e_3e_4 | V_n = \tilde{e}_1\tilde{e}_2\tilde{e}_3\tilde{e}_4) = 0 \quad \text{if} \quad e_1e_2 \neq \tilde{e}_3\tilde{e}_4.$$

Given the vectors $\boldsymbol{p}$ and $\boldsymbol{q}$ as above, the associated transition matrix P can be constructed using

$$p_{ij} = \mathbb{P}(s_i \to s_j). \tag{2.3.1}$$

The result is shown on the next page.

$$
\begin{bmatrix}
p_1q_1 & p_1(1-q_1) & (1-p_1)q_1 & (1-p_1)(1-q_1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & p_2q_3 & p_2(1-q_3) & (1-p_2)q_3 & (1-p_2)(1-q_3) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_3q_2 & p_3(1-q_2) & (1-p_3)q_2 & (1-p_3)(1-q_2) & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_4q_4 & p_4(1-q_4) & (1-p_4)q_4 & (1-p_4)(1-q_4) \\
p_5q_9 & p_5(1-q_9) & (1-p_5)q_9 & (1-p_5)(1-q_9) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & p_6q_{11} & p_6(1-q_{11}) & (1-p_6)q_{11} & (1-p_6)(1-q_{11}) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_7q_{10} & p_7(1-q_{10}) & (1-p_7)q_{10} & (1-p_7)(1-q_{10}) & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_8q_{12} & p_8(1-q_{12}) & (1-p_8)q_{12} & (1-p_8)(1-q_{12}) \\
p_9q_5 & p_9(1-q_5) & (1-p_9)q_5 & (1-p_9)(1-q_5) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & p_{10}q_7 & p_{10}(1-q_7) & (1-p_{10})q_7 & (1-p_{10})(1-q_7) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{11}q_6 & p_{11}(1-q_6) & (1-p_{11})q_6 & (1-p_{11})(1-q_6) & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{12}q_8 & p_{12}(1-q_8) & (1-p_{12})q_8 & (1-p_{12})(1-q_8) \\
p_{13}q_{13} & p_{13}(1-q_{13}) & (1-p_{13})q_{13} & (1-p_{13})(1-q_{13}) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & p_{14}q_{15} & p_{14}(1-q_{15}) & (1-p_{14})q_{15} & (1-p_{14})(1-q_{15}) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{15}q_{14} & p_{15}(1-q_{14}) & (1-p_{15})q_{14} & (1-p_{15})(1-q_{14}) & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{16}q_{16} & p_{16}(1-q_{16}) & (1-p_{16})q_{16} & (1-p_{16})(1-q_{16})
\end{bmatrix}
$$

2.3.3 Extension to memory-n case

One could extend the memory-2 IPD to a memory-n IPD by simple adjustments. We use the same adjustment as with the memory-2 IPD to make the memory-n IPD a Markov chain: we set one state of the Markov chain $\{V_n\}_{n\geq 0}$ to be the result of the first n played games. Observe that the state space S for a memory-n IPD contains $2^{2n} = 4^n$ elements. If we continue with the notation from Subsection 2.3.2, We can again set up the state space to be

$$\begin{aligned} S &= \{s_1, s_2, \ldots, s_{4^n}\} \\ &= \{CC\cdots CC, CC\cdots CD, \ldots, DD\cdots DD\}. \end{aligned}$$

The strategy vector $\boldsymbol{p}$ containing the probabilities of cooperating after observing state $s_i \in S$ also contains 4^n elements:

$$\boldsymbol{p} = (p_1, p_2, \ldots, p_{4^n-1}, p_{4^n}), \qquad q_i \in [0,1].$$

Again, the element p_i corresponds to the probability of cooperating after observing state $s_i \in S$. If we use the notation $e_1 \ldots e_{2n}$ with $e_i \in \{C, D\}$ for the states, we have

$$\mathbb{P}(V_{n+1} = e_1 \ldots e_{2n} | V_n = \tilde{e}_1 \ldots \tilde{e}_{2n}) = 0 \quad \text{if} \quad e_1 \ldots e_{2n-2} \neq \tilde{e}_3 \ldots \tilde{e}_{2n},$$

using the same reasoning as with the memory-2 IPD. We setup a transition matrix defined by $p_{ij} = \mathbb{P}(s_i \to s_j)$, and we obtain a transition matrix $P \in [0,1]^{4^n \times 4^n}$ that completely describes the memory-n IPD as a Markov chain.

When considering the computational complexity of the memory-n IPD, it is essential to take into account the structure and size of the required transition matrix P. A $4^n \times 4^n$ matrix must be created and stored to represent all possible states and transitions. The size of this matrix grows exponentially with n. Specifically, the growth rate of the matrix size is $\mathcal{O}(4^n)$.

This exponential growth has significant implications for numerical calcula-

tions. As the matrix size grows, the computational resources required — both in terms of time and memory — also increase dramatically. This leads to considerably longer processing times and greater demands on system capabilities for larger values of n. Consequently, the feasibility of performing these calculations is constrained by the available computational power.

2.4 Visualizing a Markov chain

In order to introduce definitions used in an important theorem later on, we need to consider the visualization of a Markov chain. The easiest and most intuitive way to display a Markov chain $\{V_n\}_{n\geq 0}$ with a finite number of states is by constructing a directed weighted graph $G = (S, E)$.
The set of vertices S are all the states the system can attain, so it is the state space. For any 2 elements $s_i, s_j \in S$ there is a directed weighted edge between them if the probability of transitioning from s_i to s_j is greater than zero:

$$E = \{(s_i, s_j) \mid s_i, s_j \in S \text{ and } \mathbb{P}(s_i \to s_j) > 0\}$$

The weight of an edge is given by a function $w : E \to [0, 1]$:

$$w((s_i, s_j)) = \mathbb{P}(s_i \to s_j)$$

.

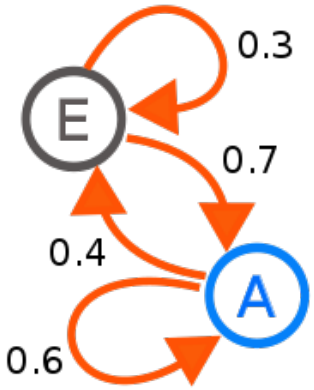

Figure 1: example of visualization of Markov chain
Source: Wikipedia

In the example above we have a Markov chain moving through 2 states. Hence, $S = \{E, A\}$. If we set state $s_1 = A$ and state $s_2 = E$, the transition matrix is given by

$$P = \begin{bmatrix} 0.6 & 0.4 \\ 0.7 & 0.3 \end{bmatrix}.$$

To help us investigate a Markov chain via a graph G, we introduce the definitions of walk, irreducibility, cycle and periodicity of a graph. For a directed weighted graph $G = (S, E)$:

Definition 2.6 (Walk). *A walk is a sequence $(s_0, s_1, \ldots, s_n)$ of consecutive (possibly repeating) vertices connected by directed edges. A walk $(s_0, \ldots, s_n)$ has length n.*

Definition 2.7 (Path). *A path is a walk that doesn't repeat any vertices, other than the first or the last vertex. More formally, a walk $s = (s_0, \ldots, s_n)$ is a path if $\forall i, j \in \{1, n-1\} : s_i \neq s_j$.*

For example, both (A, E) and (A, E, A, E, A) are valid walks in Figure 1, but only (A, E) is a path. The first walk has length 1 and the second walk has length 4.

Definition 2.8 (Irreducbility). *A graph G is irreducible if there exists a walk from any vertex to any other vertex in G.*

Definition 2.9 (Cycle). *If a path $s = (s_0, \ldots, s_n)$ has the same start and end-vertex, i.e. $s_0 = s_n$, s is a cycle.*

Definition 2.10 (Periodicity). *The periodicity of a graph G is the greatest common divisor of the lengths of all possible cycles in G. If this greatest common divisor is 1, the graph is aperiodic.*

From the definition above, we immediately remark the following:

Corollary 2.11. *Consider a directed graph G. If there is an arrow pointing from a vertex in G back to itself, i.e. a cycle of length one, then G is aperiodic.*

2.5 Limiting distributions and eigenvalues

Recall that we want to analyze the situation where every pair of players plays an infinite number of times against their opponent. It is insufficient to simply start with an initial row vector $\boldsymbol{x}_0$, and find the next n states through the transition matrix. That would only get us to a finite, n, steps. To find the results of playing an infinite number of games we introduce the definition of limiting distribution.

Definition 2.12 (Limiting distribution). *Let $\{V_n\}_{n\geq 0}$ be a Markov chain, and $S = \{s_1, \ldots, s_d\}$ its state space. Given an initial vector $\boldsymbol{x_0}$, a probability distribution $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_d)$ is called the limiting distribution of the Markov chain $\{V_n\}_{n\geq 0}$ if*

$$\pi_j = \lim_{n\to\infty} \mathbb{P}(V_n = s_j | V_0 = x_0).$$

Each element $\boldsymbol{\pi}_j$ of $\boldsymbol{\pi}$ represents the probability of the system being in state j if the process goes on infinitely. Or equivalently, it is the proportion of time the system visits state j in an infinitely iterated sequence. Since this vector is deterministic, it is also a solution of

$$\boldsymbol{\pi} = \boldsymbol{\pi} P. \tag{2.5.1}$$

Using this property, a limiting distribution vector is the left-eigenvector of P associated with eigenvalue $\lambda_1 = 1$. By looking at (2.5.1), it is clear that there exists an eigenvalue $\lambda_1 = 1$ for a transition matrix P of the memory-2 IPD. If we want to analyze long term behaviour of the IPD using the fact that $\lambda_1 = 1$ is the eigenvalue corresponding with $\boldsymbol{\pi}$, we need to make sure λ_1 is the unique dominant eigenvalue. Also, the left eigenvector $\boldsymbol{v} = (v_1, \ldots, v_d)$ associated with λ_1 needs to be a probability vector. Summarizing, the following statements have to be proven for a transition matrix P:

$$\begin{aligned} &1) \quad |\lambda_j| \leq \lambda_1 \quad \forall j \\ &2) \quad |\lambda_j| < \lambda_1 \quad \forall j \neq 1 \\ &3) \quad \sum_{i=1}^{d} v_i = 1, \quad v_i \geq 0 \quad \forall i \end{aligned} \tag{2.5.2}$$

If all these statements hold, then row eigenvector $\boldsymbol{v}$ associated with eigenvalue $\lambda_1 = 1$ is the limiting distribution $\boldsymbol{\pi}$. Moreover, $\lambda_1 = 1$ is the dominant eigenvalue, so long term behaviour is determined by $\boldsymbol{v}$. Before we begin to prove these statements, we state the Perron-Frobenius theorem:

Theorem 2.13 (**Perron-Frobenius**). *Suppose $P \in \mathbb{R}^{d\times d}$, where $p_{ij} \geq 0$, $\forall i, j$. Then, there exists a constant $z \in \mathbb{C}$ such that the eigenvector v associated with eigenvalue $\lambda_1 = 1$ has no negative elements:*

$$zv_i \geq 0, \quad i = 1, \ldots, d.$$

If we also assume that P is irreducible and aperiodic, then P has a real, dominant eigenvalue $\lambda_1 \in \mathbb{R}$ such that

$$|\lambda_j| < \lambda_1 \quad j = 2, \ldots, d.$$

It is immediately clear from the construction of transition matrix P that every element is nonnegative. However, it is not immediately obvious why this matrix is irreducible and aperiodic.

Theorem 2.14 (**Irreducibility and aperiodicity of** P). *P, as defined in (2.3.1) of Section 2.3.2, is irreducible and aperiodic.*

Proof. The Markov chain associated with the memory-2 IPD moves from state $CCCC$ back to itself with probability p_1q_1. Therefore, we have a cycle of length one. By using Corollary 2.10, we conclude that P is aperiodic.

Furthermore, suppose $G = (S, E)$, as defined in Section 2.4, is the weighted, directed graph associated with matrix P. Then, P is irreducible if and only if G is strongly connected.
Note that G is strongly connected if and only if there is a path from every state s_i to every other state s_j, where $s_i, s_j \in \{CCCC, CCCD, \ldots, DDDD\}$. Hence, it is sufficient to show that there is a path from every state s_i to every other state s_j. For simplicity, we correspond the state space S as defined in Subsection 2.3.2 to the numbers $\{1, 2, \ldots, 15, 16\}$.
By looking at the nonzero entries of the memory-2 IPD transition matrix,

we define a function f that maps subsets $I \subset S$ to other subsets $J \subset S$ if the probability of transitioning from any element $i \in I$ to any other element $j \in J$ is nonzero, i.e. $\mathbb{P}(i \to j) \neq 0, \forall i \in I, j \in J$:

$$
\begin{aligned}
f(\{1,5,9,13\}) &= \{1,2,3,4\} \\
f(\{2,6,10,14\}) &= \{5,6,7,8\} \\
f(\{3,7,11,15\}) &= \{9,10,11,12\} \\
f(\{4,8,12,16\}) &= \{13,14,15,16\}.
\end{aligned}
$$

This mapping is also shown in the figure below. Any element i can be mapped to any other element j if the column color of i is the same as the row color of j.

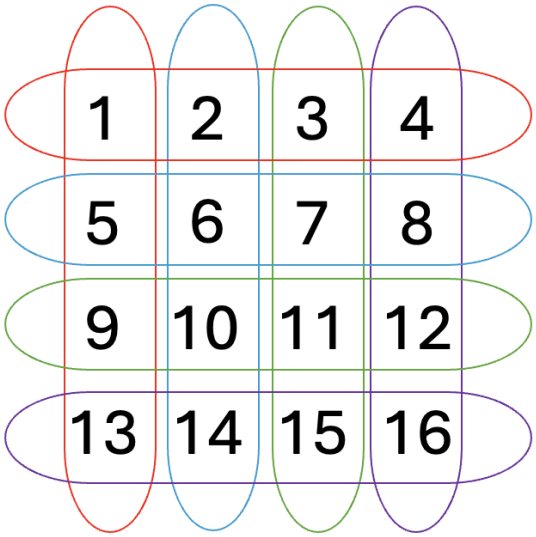

Figure 2: Mapping of function f

Now, to prove P is irreducible, we will show that it takes at most 2 transitions to go from any element $i \in S$ to any other element $j \in S$.
If the column color of i is the same as the row color of j, it takes only 1 transition to go from state i to state j. So, suppose they are not the same color. First, determine the column color of element i. In the row with the same color, find the element a that has the same column color as the row color of j. Then, the transition $i \to a \to j$ is a valid transition. Since i, j were

arbitrary, this concludes the result. □

It is important to note that this theorem does not hold when some probabilities p_i in a strategy vector $\boldsymbol{p}$ are deterministic, i.e. $p_i \in \{0, 1\}$. If, for instance, we have strategy vectors $\boldsymbol{p}$ and $\boldsymbol{q}$ where $p_1 = q_1 = 1$, then we would see that

$$p_{14} = (1 - p_1)(1 - q_1) = 0$$

We solve this by adjusting the values p_i and q_1 can attain. Instead of $p_i, q_i \in [0, 1]$, we set $p_i, q_i \in]0, 1[$. This will have minimal effect on the results, and keeps P irreducible.

Theorem 2.15. *Assume $P \in [0, 1]^{d \times d}$. Furthermore, assume $\boldsymbol{\pi}$ is the vector satisfying (2.5.1). Suppose $\lambda_1, \ldots, \lambda_d$ are the eigenvalues corresponding with matrix P, and $\boldsymbol{v}$ is the row eigenvector associated with $\lambda_1 = 1$. Then, all statements of (2.5.2) hold with $\boldsymbol{v} = \boldsymbol{\pi}$.*

Proof. 1) We define $|\boldsymbol{w}|$ for a vector $\boldsymbol{w}$ as:

$$|\boldsymbol{w}| = (|w_1|, |w_2|, \ldots, |w_d|). \tag{2.5.3}$$

With this definition, we have for any row vector $\boldsymbol{w} = (w_1, \ldots, w_d)$:

$$|\boldsymbol{w}P|_j = \left| \sum_{i=1}^{d} w_i p_{ij} \right| \leq |w_1 p_{1j}| + \ldots + |w_d p_{dj}| = |w_1| p_{1j} + \ldots + |w_d| p_{dj}$$
$$= (|\boldsymbol{w}|P)_j,$$

where the inequality follows from the triangle inequality and the p_{ij}s are nonnegative. For any left eigenvector $\tilde{\boldsymbol{v}}$ of P with eigenvalue λ, we have

$$|\lambda||\tilde{\boldsymbol{v}}| = |\lambda\tilde{\boldsymbol{v}}| = |\tilde{\boldsymbol{v}}P| \leq |\tilde{\boldsymbol{v}}|P.$$

By multiplying both sides of the inequality with $\mathbb{1} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$ on the right we obtain

$$|\lambda||\tilde{\boldsymbol{v}}|\mathbb{1} \leq |\tilde{\boldsymbol{v}}|P\mathbb{1} = |\tilde{\boldsymbol{v}}|\mathbb{1}.$$

So for any eigenvector $\tilde{\boldsymbol{v}}$ of P we have that $|\lambda||\tilde{\boldsymbol{v}}|\mathbb{1} \leq |\tilde{\boldsymbol{v}}|\mathbb{1}$, and therefore,

$|\lambda| \leq 1$ for all eigenvalues λ of P. Since $\lambda_1 = 1$, we conclude that $|\lambda_j| \leq \lambda_1$, for $j = 1, \ldots, d$.
2) By Theorem 1.13, P is irreducible and aperiodic. Hence, by the Perron-Frobenius theorem, we have that

$$|\lambda_j| < \lambda_1 \quad \forall j \neq 1.$$

3) Assume that $\boldsymbol{v}$ is the eigenvector associated with the dominant eigenvalue $\lambda_1 = 1$. According to the Perron-Frobenius theorem, there exists a constant $z \in \mathbb{C}$ such that $z\boldsymbol{v} \in \mathbb{R}$. If $\boldsymbol{v}$ is an eigenvector and $c \in \mathbb{R}$, $c \cdot \boldsymbol{v}$ is also an eigenvector. Therefore, we can set the left eigenvector $\boldsymbol{v}$ such that it is a well defined probability vector:

$$\sum_{i=1}^{d} v_i = 1, \quad v_i \geq 0 \quad \forall i.$$

□

Since a transition matrix P of the memory-2 IPD satisfies the conditions of Theorem 2.14, it satisfies (2.5.2). We can use this to determine the limiting ditribution of a transition matrix of the memory-2 IPD.

2.6 Code construction

We can use the limiting distribution to determine the payoff of 2 players playing an infinite number of games. In Section 2.5, we have proven that the limiting distribution vector is the normalized row vector $\boldsymbol{v}$ associated with the eigenvalue $\lambda_1 = 1$. We will use Python to implement the memory-2 IPD and to find dominant strategies numerically.

First, we create n players, each with a random strategy vector $\boldsymbol{p}$ as defined in Subsection 2.3.2. All elements p_i of all strategy vectors $\boldsymbol{p}$ in the population are sampled from $U(0, 1)$.
In the kth iteration, we let player i with strategy vector $\boldsymbol{p}$ play against player j with strategy vector $\boldsymbol{q}$, and determine their transition matrix P as defined by (2.3.1). We calculate the limiting distribution $\boldsymbol{\pi}$, which is the same as $\boldsymbol{v} = (v_1, \dots v_n)$, the row eigenvector associated with eigenvalue $\lambda = 1$. The element π_k of row vector $\boldsymbol{\pi}$ is the probability of the system being in state $s_k \in S$ where S is defined as in Subsection 2.3.2. If we want to determine the payoff for player i, we need to multiply $\boldsymbol{\pi}$ by the payoff column vector $\boldsymbol{p}$:

$$\boldsymbol{p} := (R, S, T, P, R, S, T, P, R, S, T, P, R, S, T, P)^T,$$

where R, S, T and P are defined as in Section 1.
We want every player of the population to play a game against every other player in the population, to ensure unbiasedness. Therefore, we construct a matrix M. The entries of M are calculated as

$$m_{ij} = \boldsymbol{p} \cdot \boldsymbol{\pi}^{(i,j)},$$

where $\boldsymbol{\pi}^{(i,j)}$ is the limiting distribution associated with the transition matrix P that was determined by the strategy vectors of player i and player j. To determine the payoff for every player in the population, we want to calculate the sum of every row of the matrix M. The sum of row i is the total payoff player i received when playing against all other players in the population. Then, we sort the players according to this total payoff.

The bottom half of the population, determined by the lowest payoffs of each

player, is omitted. The top half is retained. To replace the omitted players, new players are created. These are created by copying the retained top players, and adding noise to their strategy vectors. We define the noise $\boldsymbol{\epsilon}$ as

$$\boldsymbol{\epsilon} := (\epsilon_1, \ldots, \epsilon_{16}), \quad \epsilon_i \sim U(-0.1; 0.1).$$

The strategy vector $\tilde{\boldsymbol{p}}$ for a new player $\tilde{i}$ is then defined as

$$\tilde{\boldsymbol{p}} = \boldsymbol{p} + \boldsymbol{\epsilon},$$

where $\boldsymbol{p}$ is the strategy vector of retained player i. If this mutation results in any $\tilde{p}_i$ being outside of the open interval $]0, 1[$, we adjust it to the closest value in this interval to ensure $\tilde{\boldsymbol{p}}$ remains a well-defined strategy vector.
We then repeat the entire process, for K generations.

3 Results

In this section, we discuss the results found by running the simulation. We set $(R, S, T, P) = (3, 0, 5, 1)$ to satisfy the conditions set in the Introduction.

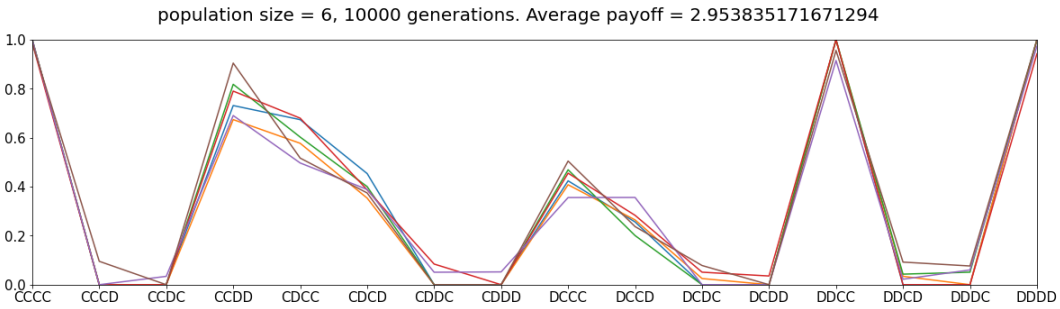

Figure 3: seed = 0

The x-axis shows all the possible outcomes of the previous two played games. Every line represents the strategy vector of a player. The line shows the probability of that player playing cooperate (C) afer observing any of the outcomes shown on the x-axis.
As expected, if a player observed 2 rounds of mutual cooperation, the next round they also choose to cooperate. Suprisingly, the probability of cooperating after 2 rounds of defection is also close to being 1. It seems that players who still defect after 2 rounds of defection receive less payoff than players who do not, and are removed from the population as a consequence of this.

In the paper '*A strategy of win-stay, lose-shift that outperforms tit-for-tat in the Prisoner's Dilemma game*' [1], the memory-1 IPD is considered. the so-called 'Pavlov strategy' showed to be the dominant strategy. The Pavlov strategy is a deterministic strategy where a player cooperates if and only if the previous round had an outcome of CC or DD. It is a simple adjustment to convert the program for a memory-2 IPD to memory-1: All that is required is to change the transition matrix to the memory-1 IPD transition matrix as defined in Subsection 2.3.1. It yields the following results:

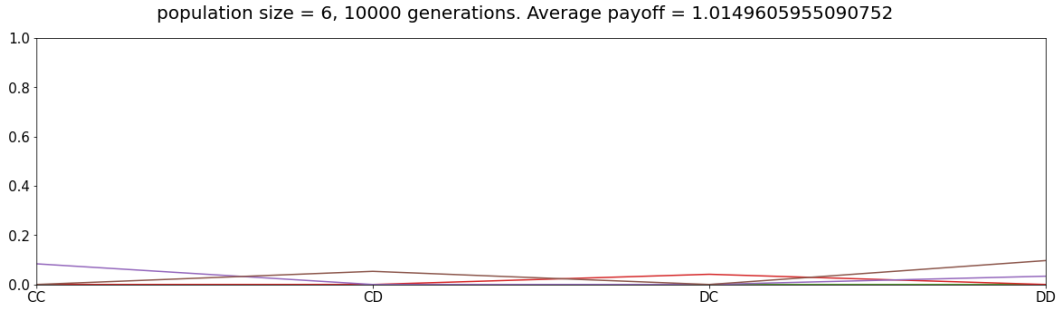

Figure 4: seed = 0

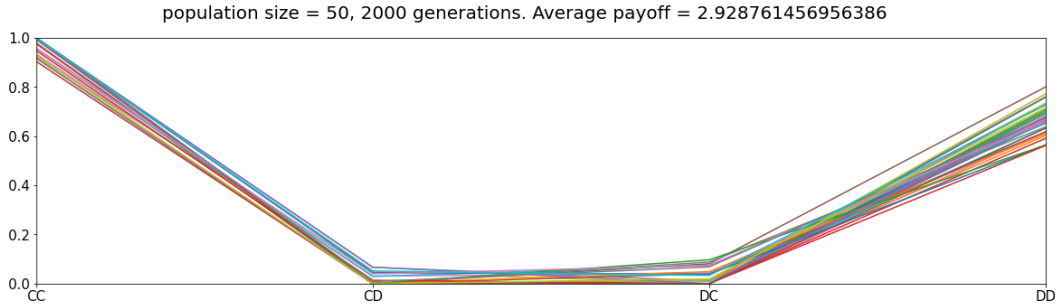

Figure 5: seed = 0

If we run the simulation with 6 players, there are no cooperators left after 100000 generations. All the players always choose defect, and therefore any new cooperators introduced into the population will immediately be taken advantage of and are omitted in the next round.

Remarkably, this is not the case if we run the simulation with the same seed, but with a larger population. It is clear that the Pavlov strategy is the dominant strategy when considering a larger population.

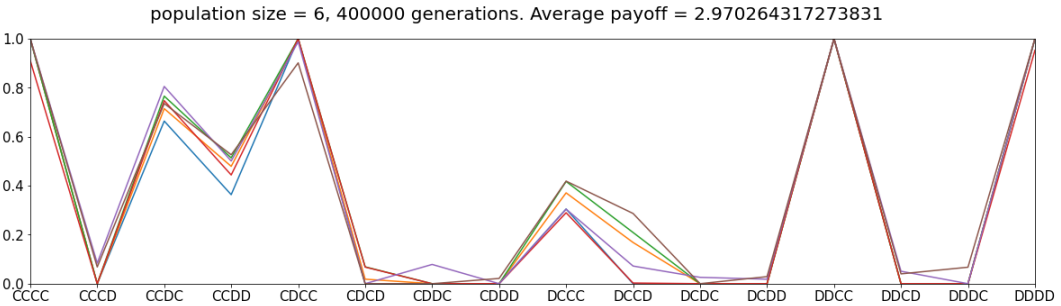

Figure 6

In the paper by M. Nowak and K. Sigmund [1], the simulations are ran for $\pm 4 \cdot 10^5$ generations. If we set the memory-2 IPD simulation to run until

$\pm 4 \cdot 10^5$ generations, the average payoff converges to 3 in the memory-2 case as well, as shown in Figure 6. The average payoff is not exactly 3 as seen in the figure. This is the result of the small adjustment made to the strategy vector when creating a new generation. An average payoff of 3 is the maximum average payoff we can attain. It is easy to determine why this is the case: since a round of CC gives both agents a reward of 3, it contributes a total of 6 to the average. 1 agent exploiting another will result in a payoff of $(5, 0)$ or $(0, 5)$, only contributing 5 to the average payoff.

After $\pm 4 \cdot 10^5$ generations, the probability of cooperating shares some characteristics with the Pavlov strategy described above: after observing either $CCCC$ or $DDDD$, the probability of cooperating is very close to 1 for all agents. The probability of cooperating is also very high for any previous states of $DDCC$ and $CDCC$, and even $CDDD$ and $DCDD$. The only difference with the Pavlov strategy is the probability for cooperating after observing an outcome of $CCDD$. Although we are unsure what is the cause of this, it is plausible that it is the result of the players having more memory than in the setup of the paper by Nowak and Sigmund [1].

It is also interesting to consider the memory-2 IPD with a large population.

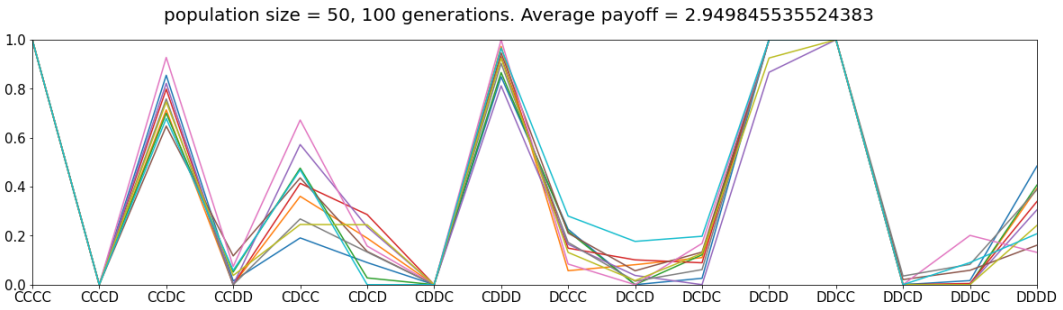

Figure 7: seed = 0

Figure 7 shows the strategies of the 10 players with the highest payoff, while Figure 8 shows the strategies of all players in the population. Surprisingly, almost all agents converge to a similar strategy in only 100 generations.

If we compare this result to Figure 3, there is a large difference in the strategies that is caused by the change in population: the probabilities for some states, such as $CCDC$ and $DCDD$ are entirely different. However, all cases

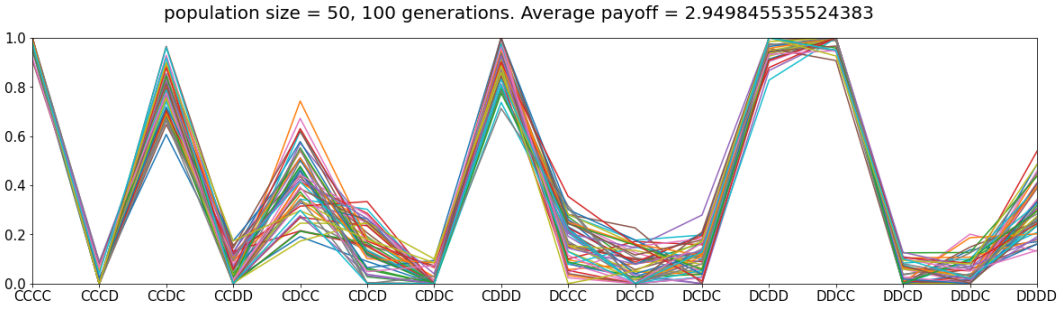

Figure 8: seed = 0

shown share one common characteristic: The probability for cooperating following a previous round of CD converges to 0. Although this cannot be seen in Figure 3, Figure 6 runs on the same seed. Therefore, it is showing the same population, where the convergence is clearly present.

In the paper *'Effects of increasing the number of players and memory size in the iterated Prisoner's dilemma: a numerical approach'* by CH. Hauert and H. G. Schuster [4], the memory-N IPD with N players is analyzed. They found that an increasing number of players and an increasing temptation T hinders the establishment of cooperation.
As seen in Figure 6, this is not the case in our results. However, in the setup of the memory-M IPD with N players, they state the following:

> Strategies, as explained previously, do not incorporate mechanisms to identify opponents. Moreover, the players cannot profit from distinguishing the opponents' moves because the encounters are set-up symmetrically [...]. Thus, the probability to cooperate depends only on the number of cooperating respectively defecting opponents in each round. This reduces the strategy space dramatically from $2^M = 2^{mN}$ to 2^{mN} dimensions, where $m \in \mathbb{N}$ corresponds to the number of rounds recalled by the players. Therefore, for example, for three players with three-step memory the following identities must hold: $p_{ccd} \equiv p_{cdc}$ and $p_{dcd} \equiv p_{ddc}$. [4]

If the IPD is set up as above, it indicates that players cannot differ between when an opponent defected: the probability of cooperating after re-

sults *CDCC* and *CCCD* would be identical. The author does this to reduce the strategy space and thereby CPU time. However, it removes the ability of recognizing whether an opponent is starting to defect after cooperating, or whether an opponent is starting to cooperate after previously defecting. This could be the cause of the difference in the results. To verify this, we compute another run with an even higher population size.

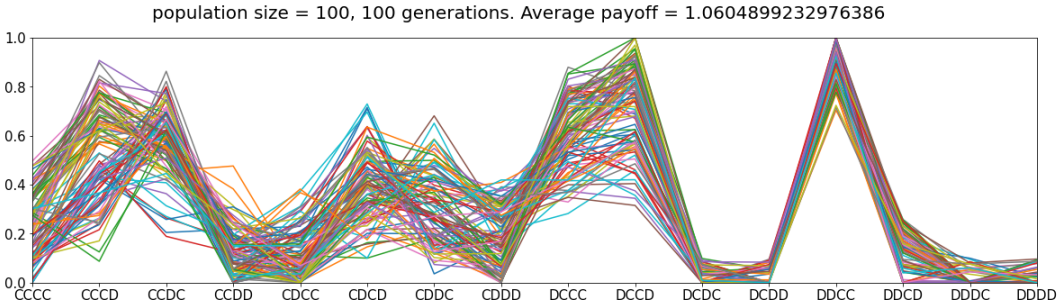

Figure 9: seed = 1

As shown by the average payoff for this population, there is a trend of lower tendency to cooperate, and therefore there is more defection. But this is not as obvious as described in the paper [4].

4 Conclusion and future work

In this study, we investigated the effects of increasing memory size and population in the Iterated Prisoner's Dilemma (IPD). Our results provide insights into how these factors influence the emergence and stability of cooperative strategies.

Increasing the memory size of agents in the IPD substantially impacts their strategic behaviors. With larger memory, agents can recall and condition their actions on a longer history of past interactions, which generally leads to more nuanced and sophisticated strategies. This increased memory allows for a better understanding of the opponent's behavior over multiple rounds, thereby enhancing the ability to predict and respond to future actions. For instance, with memory-2, agents show a higher probability of cooperation after observing mutually cooperative or defective rounds, aligning with the Pavlov strategy, which is known for its robustness and effectiveness.

The dominant strategies that emerge under these conditions tend to share several characteristics. Firstly, they often exhibit a strong conditional response to past interactions, particularly favoring cooperation if previous rounds resulted in mutual cooperation or mutual defection. This is evident in the high probability of cooperation following such outcomes. Secondly, there is a convergence towards specific probabilistic strategies, where almost all agents in the population adopt similar tactics within relatively few generations, indicating a form of strategic equilibrium. Moreover, dominant strategies typically involve a mechanism to discourage exploitation; for example, the probability of cooperation after encountering a defection tends to be low, which helps in maintaining stability against defectors.

In conclusion, the expansion of memory in agents within the IPD framework fosters the development of more sophisticated and cooperative strategies. The emergent dominant strategies share the characteristics of conditional cooperation and convergence, contributing to a stable cooperative environment even as the complexity of the system increases.

Future research could further explore the effects of larger memory sizes and larger population sizes on the dynamics of the IPD. While our study focused on memory-2, examining memory-3 or even larger memory sizes could provide deeper insights into strategic development and stability. Additionally, increasing the population size could reveal more about cooperative behavior and the robustness of the dominant strategies identified in smaller populations.

We are currently limited by computational power, which constrains our ability to simulate larger memory sizes and populations. Advances in computational resources or more efficient algorithms could facilitate these studies, enabling a deeper exploration of how a larger scale affects the evolution of cooperation. Investigating these aspects may help in understanding how cooperation can be maintained in more complex and larger-scale systems, providing broader applications in fields such as economics, political science, and evolutionary biology.

References

[1] M. Nowak and K. Sigmund, "A Strategy of Win-Stay, Lose-Shift That Outperforms Tit-for-Tat in the Prisoner's Dilemma game" *Nature*, vol 364, 56–58, 1993.

[2] S. Li, "Strategies in the Stochastic Iterated Prisoner's Dillema," 2014.

[3] M. Ueda, "memory-two zero-determinant strategies in repeated games" *R. Soc. Open Sci.*, `https://doi.org/10.1098/rsos.202186`

[4] CH. Hauert and H. G. Schuster, "Effects of increasing the number of players and memory size in the iterated Prisoner's dillema: a numerical approach" *Proc. R. Soc. Lond. B* vol 264, 513–519, 1997.

Appendix

Python code for memory-2 IPD

Below is the complete code for calculating the figures associated with the memory-2 IPD, presented in Section 3.

```
import copy
import numpy as np
import scipy.linalg as sclin
import matplotlib.pyplot as plt

n = 50 # population size
K = 20 # amount of generations
np.random.seed(0) # save random seed

class Prisoner:
    def __init__(self,strategy,payoff = 1):
        self.strategy = strategy
        self.payoff = payoff

def transition_matrix(p,q): # p, q are 16-tuples of strategies
 ↪  player 1 & 2
    def f(pi,qi):
        return [ pi * qi, pi * (1-qi), qi * (1-pi), (1-pi) *
         ↪  (1-qi) ]

    P = np.zeros((16,16))
    P[0,    :4 ] = f(p[0 ],q[0 ])
    P[1,   4:8 ] = f(p[1 ],q[2 ])
    P[2,   8:12] = f(p[2 ],q[1 ])
    P[3,  12:  ] = f(p[3 ],q[3 ])

    P[4,    :4 ] = f(p[4 ],q[8 ])
```

```
26      P[5,   4:8 ] = f(p[5 ],q[10])
27      P[6,   8:12] = f(p[6 ],q[9 ])
28      P[7,  12:  ] = f(p[7 ],q[11])
29
30      P[8,    :4 ] = f(p[8 ],q[4 ])
31      P[9,   4:8 ] = f(p[9 ],q[6 ])
32      P[10,  8:12] = f(p[10],q[5 ])
33      P[11, 12:  ] = f(p[11],q[7 ])
34
35      P[12,   :4 ] = f(p[12],q[12])
36      P[13,  4:8 ] = f(p[13],q[14])
37      P[14,  8:12] = f(p[14],q[13])
38      P[15, 12:  ] = f(p[15],q[15])
39
40      return  P
41
42  def avgpay(player_1, player_2):
43
44      # calculates average payoff for player 1, where player 1 vs
        ↪  player 2
45
46      trmatrix =
        ↪  transition_matrix(player_1.strategy,player_2.strategy)
47      eigen,vl = sclin.eig(trmatrix, left = True, right = False)
48
49      index = np.where(np.isclose(eigen, 1, atol = 1e-8))[0][0]
50      # find index of eigenvalue equal to 1
51
52      eigenvector = np.real(np.transpose(vl)[index])
53      eigenvector = (1 / sum(eigenvector))*eigenvector #
        ↪  normalise eigenvector
54
```

```
55      return np.inner([eigenvector],4*PAYOFF)
56
57  def clip(l):
58      # list as input, to keep strategies welldefined and the
        ↪  transition matrix
59      # irreducible
60      for x in range(len(l)):
61          if l[x] < 1e-4:
62              l[x] = 1e-4
63          elif l[x] > 1-1e-4:
64              l[x] = 1-1e-4
65          else:
66              pass
67      return l
68
69  def play_matrix(n):
70      # n = population size, matrix where Mij = game player i vs.
        ↪  player  j
71
72      M = np.zeros((n,n))
73      for i in range(n):
74          for j in range(n):
75              M[i,j] = avgpay(PLAYERS[i],PLAYERS[j])
76
77      return M
78
79  def average(l):
80      return sum(l)/len(l)
81
82  plt.rc('axes', labelsize=15)    # fontsize of the x and y
    ↪  labels
83  plt.rc('xtick', labelsize=15)   # fontsize of the tick labels
```

```
84  plt.rc('ytick', labelsize=15)   # fontsize of the tick labels
85  plt.rc('figure', titlesize=20)  # fontsize of the figure title
86  z = np.arange(16) #x-axis of graph
87
88  order =
    ↪  ['CCCC','CCCD','CCDC','CCDD','CDCC','CDCD','CDDC','CDDD','DCCC','DCCD','DCDC
89  PAYOFF = [3,0,5,1]
90
91  strategies = np.random.uniform(0,1,(n,16)) #generate different
    ↪  random strategies
92
93  PLAYERS = []
94  PLAYERS += [Prisoner(x) for x in strategies]
95
96  for h in range(1,K+1): # loop generations
97      game = play_matrix(n)
98      payoffs = np.sum(game, axis = 1)
99
100     for i in range(len(PLAYERS)):
101         PLAYERS[i].payoff  = payoffs[i]
102
103     PLAYERS.sort(key=lambda x: x.payoff, reverse=True)
104     # sort players according to payoff
105
106     PLAYERS = PLAYERS[:len(PLAYERS)//2]
107     # omit worst half
108
109     NEW_PLAYERS = copy.deepcopy(PLAYERS)
110
111     noise = np.random.uniform(-0.1,0.1,(len(NEW_PLAYERS),16))
112     # add mutation to strategy of new players
113
```

```
114     for i in range(len(NEW_PLAYERS)):
115         NEW_PLAYERS[i].strategy = clip(NEW_PLAYERS[i].strategy
            ↪  + noise[i])
116
117     PLAYERS += NEW_PLAYERS
118
119     if h%1 == 0:
120         f = plt.figure()
121         f.set_figwidth(20)
122         f.set_figheight(5)
123         for i in range(len(PLAYERS)):
124         #for i in range(10): #for showing best 10 players only,
            ↪  for large pop. size
125
126             plt.xticks(z,order)
127             plt.plot(z,PLAYERS[i].strategy)
128             plt.autoscale(False)
129             f.suptitle('population size = '+str(n)+',
                ↪  '+str(h)+' generations. Average payoff =
                ↪  '+str(average(payoffs)/n) )
130         plt.show()
131
```