Thesis Title

Thesis Subtitle

Author Name

B.Sc. Final Year Dissertation

Cardiff School of Mathematics

Acknowledgments

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Chapter 2

Literature Review

The Prisoner's Dilemma is a very popular model in game theory and there have been many papers written about the subject. The game has been applied to many different research areas is often used to model systems in biology [34], sociology [13], psychology [17], and economics [9]. The start of this chapter will give a brief overview of the literature and particularly relevant work will be highlighted. This is followed by an outline of how Axelrod's work is currently being reproduced by an open-source community. Finally, an introduction to fingerprinting and some necessary definitions and theorems are given at the end of the chapter.

2.1 Background

The political scientist Robert Axelrod held the first IPD tournament in 1980 [3]. Many well-known game theorists were invited to submit strategies that would compete against each other in a round robin style format. All strategies also competed against a random strategy (that would randomly choose between 'C' and 'D') and a copy of themselves. All strategies knew that the length of each game was 200 moves, and the whole tournament was repeated 5 times for reliability. Out of the 13 strategies that were entered, TitForTat was announced as the winner and was submitted by Professor Anatol Rapoport from the Department of Psychology of the University of Toronto.

TitForTat is a very simple strategy (see Section 2.2.1) and as explained in [4], it won because of three defining characteristics:

- 'Niceness' - A strategy is said to be nice if it is not the first to defect.
- 'Provocability' - Immediately after an opponent defects, the strategy should defect in retaliation.
- 'Forgiveness' - The strategy is willing to continue with mutual cooperation even after some defections.

Axelrod's second tournament [4] saw a dramatic increase in terms of size, with 62 strategies being entered from 6 different countries. The contestants ranged from a 10-year-old computer hobbyist to professors of computer science, economics, psychology, mathematics, sociology, political science and evolutionary biology. The countries represented were the United States, Canada, Great Britain, Norway, Switzerland, and New Zealand. Despite the fact that all contestants had full knowledge of the previous tournament, TitForTat was the overall winner once again. One large difference in the mechanics of the first and second tournament was that the second tournament did not specify how many moves a game would last. Instead, the game ended probabilistically with a 0.00346 chance of finishing on any given move. This parameter was chosen so that the median length of a game would be 200 moves (in line with the first tournament).

Year	Reference	Number of Strategies	Type
1979	[3]	13	Standard
1979	[4]	64	Standard
1984	[6]	64	Evolutionary
1991	[8]	13	Noisy
2005	[10]	223	Varied
2012	[36]	13	Standard

Table 2.1: An overview of published tournaments

Axelrod continued to extend his work by considering an evolutionary version of the tournament [6, 5]. In this case, the proportion of the population playing a certain strategy depends on how the strategy performed on the previous round. A strategy is evolutionarily stable if a population of individuals using that strategy cannot be invaded by a rare mutant adopting a different strategy [6].

In [28, 27, 29], Nowak extends this further by studying Spatial Games. In his variation, the game is played on a 2 dimensional square lattice where the payoff for an individual is the sum over all interactions with its 8 nearest neighbours (see Figure 2.1) and itself. In the next generation, an individual cell is occupied with the strategy that received the highest payoff among all 8 nearest neighbours and itself.

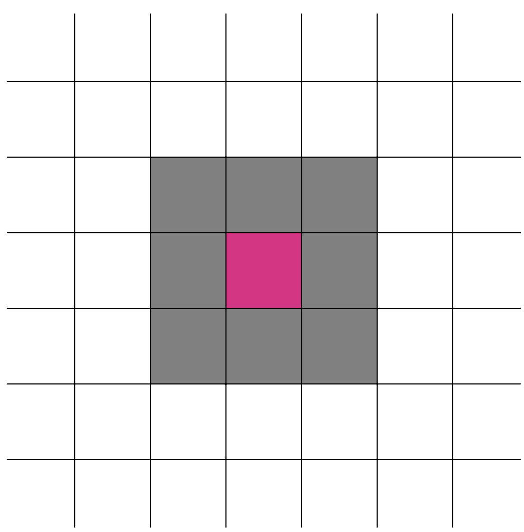


Figure 2.1: The topology of Nowak's spatial variation

To simplify things, Nowak limited each cell to be either Cooperator or Defector. Thus the game is completely deterministic and the outcome depends only on the initial configuration and the payoff matrix. One particularly interesting result is that if a single Defector invades a world of Cooperators, long (but finite) sequences of patterns emerge. Due to the symmetry of the game rules, all the patterns are highly symmetric and have the appearance of kaleidoscopes.

The primary purpose of [8] was to discover the effect of noise on TitForTat's performance in IPD. As had been previously claimed its performance was significantly reduced. The authors of [8] suggest that this is due to the provocability mentioned earlier, which in the presence of noise, can causes it to fall into unintended vendettas with other nice, but provocable strategies. Strategies that outperformed TitForTat were far more forgiving, which enabled them to get back to mutual cooperation much faster after an accidental defection.

2.2 Strategies of Particular Interest

Throughout this report many different strategies will be discussed and used in small examples. This section will provide a description of some of these strategies in order to aid the reader, as a strategy's name is not always particularly descriptive.

2.2.1 TitForTat

TitForTat is the most well known strategy for playing Prisoner's Dilemma due to it winning both of Axelrod's first two tournaments. It starts with a cooperative move and proceeds to play the same as the opponent did on the previous move [4, 14].

2.2.2 Pavlov

In [27], the strategy Pavlov is introduced (sometimes referred to as Win-Stay Lose-Shift). Pavlov plays by repeating its previous move if it was successful (received payoff T or R), and swapping the move if it was unsuccessful (received payoff P or S). The paper explains how this allows it take advantage of strategies that cooperate unconditionally and can also correct occasional mistakes.

2.2.3 Gradual

The strategy Gradual is present in [7]. It begins the game by cooperating, then after the first defection of the other player, it defects one time and cooperates twice. After the second defection of the opponent, it defects two times and cooperates twice. After the n^{th} defection it reacts with n consecutive defections and then two cooperations. In [7] it is claimed that Gradual outperforms TitForTat and this has been confirmed by work done through the Axelrod-Python library [22].

2.2.4 Random

Random plays exactly as expected, by choosing randomly between whether to cooperate or defect. The probabilities of choosing cooperate or defect are not necessarily even, instead they could rely on some distribution. We will denote the strategy that chooses to cooperate with probability p as Random(p). For example, the strategy that randomly chooses to cooperate 20% of the time and defect 80% of the time would be Random(0.2).

2.2.5 Cooperator/Defector

Cooperator and Defector are very simple. Cooperator will choose to cooperate at every turn, and similarly Defector will choose to defect every turn.

2.2.6 Cycler

The strategy Cycler(s) will repeat a given sequence of moves s. For example Cycler(CD) would play CDCDCD... and Cycler(CDDDC) would play CDDDCCDDDCCDDDC...

2.2.7 Evolved Looker-Up

The winning strategy in the version of Axelrod-Python used throughout this paper is Evolved Looker-Up. The author, Martin Jones, has produced a thorough explanation of how the strategy was written in [18], and a brief outline will be given here. Quite simply, the strategy uses a lookup table to determine the action it should take. A lookup table is similar to a dictionary when programming, where the keys could be the opponents previous moves and the values the next action the strategy should take. The length of the key could be extended to include the opponents two previous moves (or more). It is clear that for a key length l, there are $n = 2^l$ keys. For each key the next action could be Cooperate or Defect, and so there are a total of 2^n possible lookup tables. Evolved Looker-up uses as its keys: the opponents first two moves, the opponents previous two moves, and the strategies own previous two moves. So we have $n = 2^6 = 64$ keys, and therefore $2^{64} \simeq 10^{18}$ possible lookup tables.

An evolutionary algorithm is then used to find the best possible table. The operates by initially creating a population of many different lookup tables. Then a new lookup table is created by combining two pre existing ones together (similar to how DNA is joined in reproduction). There is a probability (normally low) of some of the new lookup table mutating. Next, several more random lookup tables are produced. Finally, all lookup tables are scored and poor performers are discarded. Then the whole process is repeated with the new generation of lookup tables.

Evolved Looker-Up was produced after 200 generations, and the final lookup table can be found in the Axelrod-Python source code. Alternatively, the code used to produce the lookup table is given in [24].

2.3 Finite State Machines and Automaton

- what has already been done - how do they relate to this project

An Automaton is an abstract model of a machine that can perform operations on an input. A more general (and powerful) abstraction of an automaton is the famous Turing machine, which can perform any computational process carried out by present day computers [21]. In the case where an automaton has a finite set of set of states, it is referred to as a Finite State Machine. If this FSM has outputs it is called a Moore Machine [26]. A more detailed of Finite State Machines is given in Section 3.5.

Finite state machines have become important in game theory for several reasons. In the late fifties the idea of "bounded rationality" was explored in economics by Simon [35]. This was then extended in [32] where strategies that played the IPD were implemented as Moore Machines with an extra rule that added a cost associated with the complexity of the Moore Machine. Complexity was defined to be the number of states in the machine and the author states that this was a "fairly naive" approach.

In [19, 20], proofs are given that show that every strategy can be represented by an automaton. However, in Section 3.6 it is shown that if the length of a game in IPD is known, every strategy can be represented as a FSM.

2.4 Axelrod-Python Library

The Axelrod-Python library [12] is an open source Python package for creating reproducible research into the Prisoner's Dilemma. The original aim was to recreate Axelrod's tournaments as described in section 2.1 and verify their results. As the library has grown, so has the overall aim. The goal now, is "to provide a resource, with facilities for the design of new strategies and interactions between them, as well as conducting tournaments and ecological simulations for populations of strategies" [22].

For many of the tournaments that have been described the original source code is not available, and in the few cases where access was available there were no tests and minimal documentation. The library is partly motivated by a desire to improve this situation, and there is much discussion within academia currently regarding reproducible research [11, 15, 30, 33]. Some key characteristics are that the library is:

- Open: all code is released under an MIT license [31]
- Reproducible and well-tested: at the time of writing there is an excellent level of integrated tests with 99.73% coverage (including property based tests: [23])
- Well-documented: all features of the library are documented for ease of use and modification
- Extensive: 139 strategies are included, with infinitely many available in the case of parametrised strategies
- Extensible: easy to modify to include new strategies and to run new tournaments

Each of these items will be discussed in more detail in Section 4.5.

Figure 2.2: Ranked violin plot of the mean payoff for each player

Figure 2.3: Matrix plot of pair wise payoffs for each player

In listing 1 an example of how to produce a simple tournament is shown. Lines 7 - 10 create two plots. The first is a ranked violin plot of the mean payoff for each player and the second is a matrix plot of pair wise payoffs for each player. These plots can be seen in figures 2.2 and 2.3.

```
1  >>> import axelrod as axl
2  >>> axl.seed(0)  # Set a seed
3  >>> players = [s() for s in axl.strategies]  # Create players
4  >>> tournament = axl.Tournament(players)  # Create a tournament
5  >>> results = tournament.play()  # Play the tournament
6  >>> plot = axl.Plot(results)
7  >>> p = plot.boxplot()
8  >>> p.show()
9  >>> q = plot.payoff()
10 >>> q.show()
```

Listing 1: Example code to produce a simple tournament

2.5 Fingerprinting

It is easy to write a genetic algorithm to produce large numbers of strategies, however their analysis can be time consuming. Even the simple question 'Are these two strategies the same?' does not always have an obvious answer. This is especially true when considering source code, as two identical strategies can be coded in different ways. For example, differentiating between Random(0.5) and Cycler(CD) (see Section 2.2) isn't trivial unless you have access to their source code. The results of a game they play isn't necessarily enough.

A method for comparing strategies is first given in [2]. Ashlock outlines several definitions, theorems and proofs concerning the construction of a fingerprint. These are then followed by some examples, however they are of low quality and the only probe (see definition 4) used is TitForTat.

Ashlock then extends his fingerprinting work further in [1]. More examples are presented, and many different fingerprint functions are listed. Also, a large number of the analytical fingerprint functions use a probe that is not TitForTat. Fingerprinting is then used to to assess how three evolutionary algorithms produce different populations. The evolutionary methods involved are finite-state machines, lookup tables and feed forward neural nets.

Chapter 3

Theory

Many definitions will be now be presented, with the overall aim being to have a rigorous definition for a fingerprint. Definitions for the Dual, Joss-Ann and Fingerprint were first presented by Ashlock in [2] as mentioned in Section 2.5. Then a definition of Finite State Machines will be given, and an explanation of how they relate to fingerprinting. Finally, an example of how to construct a fingerprint will be shown in Section 3.7.

3.1 The Joss-Ann

The Joss-Ann is a basic transformation that can be applied to a strategy. It operates by making a probabilistic choice of cooperation, defection or the original move. More formally:

Definition 1 *If A is a strategy for playing the iterated prisoner's dilemma, then the **Joss-Anne of A**, $JA(A, x, y)$ is a transformation of that strategy. Instead of the original behaviour, it makes move 'C' with probability x, move 'D' with probability y, and otherwise uses the response appropriate to strategy A (if $x + y < 1$).*

The notation JA comes from the initials of the names Joss and Anne. Joss was a strategy submitted to one of Axelrods original tournaments and it would occasionally defect without provocation in the hopes of a slight improvement in score. Anne is the first name of A. Stanley who suggested the addition of random cooperation instead of random defection [1]. When $x + y = 1$, the original strategy is not used, and the resulting behaviour is a random strategy with probabilities (x, y). In more general terms, a JA strategy is an alteration of a strategy A that causes the strategy to be played with random noise inserted into the responses.

3.2 The Dual

The Dual is another, more complex transformation of a strategy (note that transformations can be applied over each other). Given a history for an opponent, the responses of the original strategy and the dual would be opposite.

Definition 2 *Strategy A' is said to be the* ***Dual*** *of strategy A if A and A' can be written as finite-state machines that are identical except that their responses are reversed.*

It is important to note that this is different to taking a strategy and flipping it's responses. The dual relies on knowledge of the underlying state of the original strategy, whereas the flip does not. This is shown in Table 3.1. For an outline of how Pavlov plays, see Section 2.2.2.

3.3 The Flip

The Flip is a simple transformation that returns the opposite of the strategy. This is subtly different from the Dual as demonstrated below.

Definition 3 *Strategy A' is said to be the* ***Flip*** *of strategy A if, A and A' return different actions for the same game history (includes opponent and self).*

Opponent	Pavlov	Dual	Flip
C	C	D	D
D	C	D	C
D	D	C	C
C	C	D	C
C	C	D	D
D	C	D	C
C	D	C	C
D	D	C	D
	C	D	D

Table 3.1: The different responses of Pavlov, Pavlov's Dual and Flipped Pavlov

The subtle difference between Dual and Flip can be highlighted further by inspecting each row individually.

Row 1 - Pavlov always plays 'C' on the first go. Flip will change this to 'D'. Dual knows that Pavlov always plays 'C' and so swaps to 'D'.

Row 2 - In the previous round for Pavlov the strategies played (C, C), and so Pavlov plays 'C' again. For Flip, the preceding interaction was (D, C), in this instance Pavlov would play 'D' again, so this gets flipped to 'C'. The previous turn for Dual was (D, C) so it infers that Pavlov had (C, C). It knows that Pavlov would play 'C' and so plays 'D'.

Row 3 - In the previous round for Pavlov the strategies played (C, D), and so Pavlov would change to play 'D'. For Flip, the preceding interaction was (C, D), in this instance Pavlov would change to 'D', so this gets flipped to play 'C' again. The previous turn for Dual was (D, D) so it infers that Pavlov had (C, D). It knows that Pavlov would play 'D' in this instance and so plays 'C'.

3.4 Fingerprint and Double Fingerprint

The fingerprint function returns the expected score of a strategy when it plays against the Joss-Ann with varying (x, y). The double fingerprint extends this idea to $x + y \geq 1$.

Definition 4 *A **Fingerprint** $F_A(S, x, y)$ with $0 \leq x, y \leq 1$, $x + y \leq 1$ for strategy S and **probe** A, is the function that returns the expected score of strategy S against $JA(A, x, y)$ for each possible (x, y).*

Definition 5 *The **Double Fingerprint** $F_{AB}(S, x, y)$ with $0 \leq x, y \leq 1$ returns the expected score of strategy S against $JA(A, x, y)$ if $x + y \leq 1$, and $JA(B, 1 - y, 1 - x)$ if $x + y \geq 1$.*

We now show how the double fingerprint, with an appropriate choice of probe, is equivalent to the fingerprint function over the unit square.

Theorem 1 *If A and A' are dual strategies, then $F_{AA'}(S, x, y)$ is identical to the function $F_A(S, x, y)$ extended over the unit square.*

A proof for this theorem will now be given which was first presented in [2]:

Proof 1 *The Markov chain for the dual strategy A' will have the same transitions as the Markov chain for the strategy A. However, each entry for x corresponds to the probability that the strategy $JA(A, x, y)$ will randomly choose 'C' when it would not normally do so. For strategy A', this will occur whenever $JA(A', x, y)$ does not randomly respond 'D', which has probability $1 - y$.*

Similarly, each y corresponds to the probability that the strategy $JA(A, x, y)$ will randomly choose 'D' when it would usually respond 'C'. For strategy A', this will occur whenever $JA(A', x, y)$ does not randomly respond 'C', which has probability $1 - x$.

Thus the Markov chain for $JA(A', x, y)$ is the Markov chain for $JA(A, x, y)$ with the mapping $(x, y) \to (1 - y, 1 - x)$. Therefore $F_{AA'}(S, x, y)$ extends to the remainder of the unit square the function given by $F_A(S, x, y)$. ■

Theorem 1 allows the fingerprint function to naturally extend over the unit square. Figure 3.1 shows that in the bottom left region of the plot, the strategy plays against JA(A, x, y) and in the top right region it plays against JA$(A', 1 - y, 1 - x)$.

3.5 Finite State Machines

A formal definition of a Finite State Machine is given by Definition 6 but first we will outline some motivating key characteristics of a system that can be modelled with a FSM:

Figure 3.1: Whether the Strategy is probed by the Dual or not

- The system must be describable by a finite set of states.
- The system must have a finite set of inputs that can trigger transitions between states.
- The behaviour of the system at a given point in time depends upon the current state and the input that occurs at that time.
- For each state the system may be in, behaviour is defined for each possible input.
- The system has a particular initial state.

We can make the above bullet points rigorous for deterministic cases with the following definition:

Definition 6 *A **Deterministic Finite State Machine** M is a tuple $(S, \sigma, \delta, s_0, F)$ where*

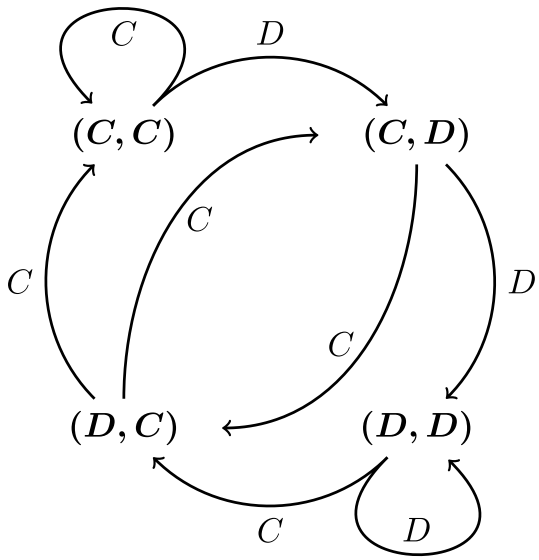
- *σ is the set of symbols representing the input of M.*
- *S is the set of states of M.*
- *$s_0 \in S$ is the starting state.*
- *$F \subseteq S$ is the set of final states of M.*
- *$\delta : S \times \sigma \to S$ is the transition function.*

Figure 3.2a and figure 3.2b show FSM representations for TitForTat and Pavlov respectively (see Sections 2.2.1 and 2.2.2 for an explanation of how these strategies operate). Here nodes represent the previous action taken by the strategy and the opponent, ie. node (D, C) implies that on the preceding turn, the strategy chose to Defect and the opponent chose to Co-operate. Arcs represent the choice made by the opponent at the current turn, and lead us to the state for the next turn.

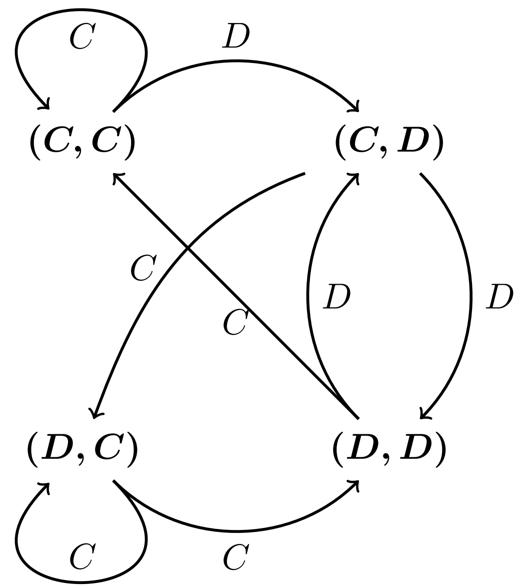
These are not necessarily the simplest FSM representation of the strategies. For example, TitForTat requires no knowledge of its own previous moves, but they have been included for completeness.

In figure 3.2c we have a more complex FSM for the strategy Majority for a game with 4 turns. Majority plays in the following way:

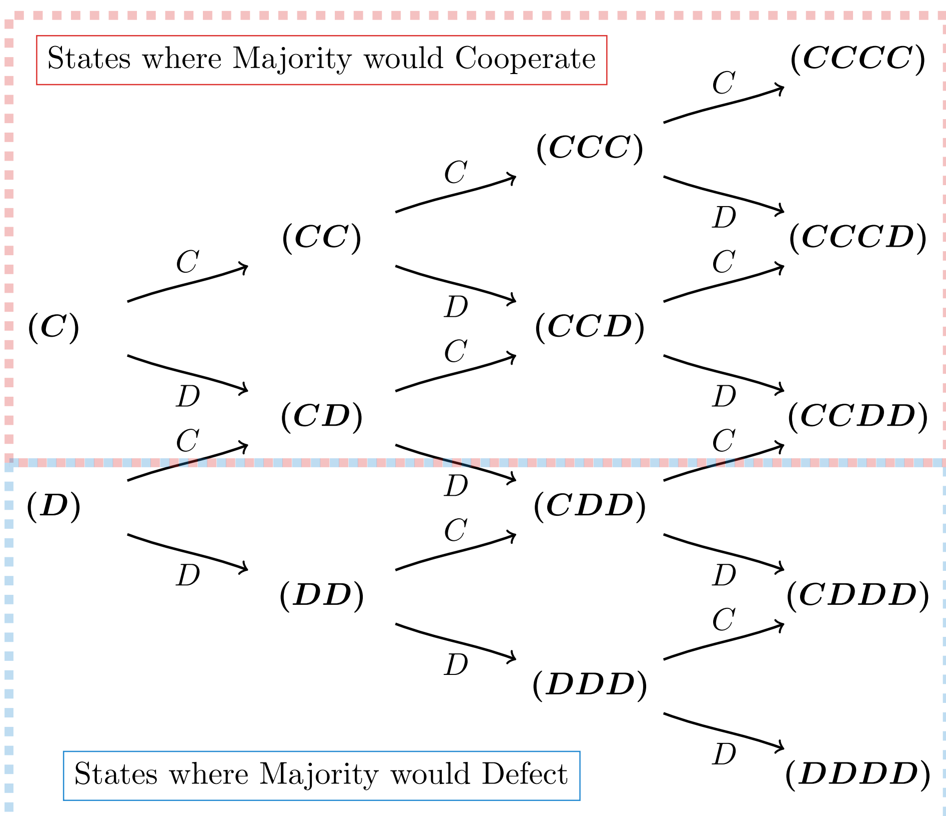
- If the opponent has cooperated the majority of the time, Majority will cooperate
- If the opponent has defected the majority of the time, Majority will defect



(a) FSM for TitForTat



(b) FSM for Pavlov (Win-Stay Lose-Shift)



(c) FSM for Majority in a game with 4 Turns

Figure 3.2: Finite State Machine representations for TitForTat, Pavlov and Majority

- Note - the strategy shown is technically Soft Majority, if the opponents cooperations and defections are equal it will cooperate. Hard Majority would defect in this situation.

Clearly this means that the strategy Majority requires knowledge of all previous states. In general this implies that Majority could not be represented as a FSM. However if the number of turns in a game is known, any strategy can be represented as a FSM.

3.6 Proof of FSM for every strategy

3.7 Analytical Fingerprints

There are several steps to constructing the Fingerprint of a strategy and basic knowledge of Markov Chains is required. An outline of the steps is as follows:

1. Build a Markov chain model of an IPD between the strategy and probe strategy.
2. Construct the corresponding transition matrix.
3. Find the steady state distribution.
4. Calculate the overall expected score by taking the dot product of the steady state distribution with the payoff vector given in section to obtain the fingerprint function.
5. This can then be plotted as a heat map to make it easier to visualize.

As an example, this process will now be applied to obtain a fingerprint for the strategy Win-Stay-Lose-Shift (sometimes referred to as Pavlov) when probed by Tit-For-Tat.

Step 1 - Build the markov chain.

Step 2 - Construct the transition matrix.

$$T = \begin{array}{c} \\ (C,C) \\ (C,D) \\ (D,C) \\ (D,D) \end{array} \begin{array}{c} \begin{array}{cccc} (C,C) & (C,D) & (D,C) & (D,D) \end{array} \\ \begin{pmatrix} 1-y & 0 & 0 & x \\ y & 0 & 0 & 1-x \\ 0 & 1-y & x & 0 \\ 0 & y & 1-x & 0 \end{pmatrix} \end{array} \tag{3.1}$$

Step 3 - Find the steady state distribution.

$$\pi = \begin{bmatrix} \frac{x(1-x)}{2y(1-x)+x(1-x)+y(1-y)}, \\ \frac{y(1-x)}{2y(1-x)+x(1-x)+y(1-y)}, \\ \frac{y(1-y)}{2y(1-x)+x(1-x)+y(1-y)}, \\ \frac{y(1-x)}{2y(1-x)+x(1-x)+y(1-y)} \end{bmatrix} \tag{3.2}$$

Step 4 - Calculate the expected score.

$$F = \pi \cdot \begin{bmatrix} 3 \\ 0 \\ 5 \\ 1 \end{bmatrix} = \frac{3x(1-x)+y(1-x)+5y(1-y)}{2y(1-x)+x(1-x)+y(1-y)} \tag{3.3}$$

Step 5 - Plot the resulting function.

Theorem 2 *Given a deterministic strategy α and 2 histories h_1, h_2, then for all games of length $n \in \mathbb{N}$ there exists a FSM such that $\alpha(h_1, h_2)$ can be obtained from the FSM.*

Proof 2 *Let $\sigma = \{C, D\}$ and*

$$S = \bigcup_{i=0}^{n+1} \{C, D\}^i \times \{C, D\}^i \delta((h_1, h_2), a) = ()$$

Chapter 4

Implementation

This chapter will explain how the method of fingerprinting was implemented within the Axelrod Python Library. Each of the definitions presented in Chapter 3 directly correspond to functions which are described below. All strategies have an equivalent class within the library and this is demonstrated in Section 4.3.

4.1 The Dual

The dual of a strategy is defined such that when the original strategy and the dual are presented with identical histories they will return opposite actions, as outlined in Definition 2. This means the dual relies on knowledge of how the original strategy would have behaved in a given situation, which is impractical to infer from the source code. However, the required behaviour can be achieved by having the original strategy as an attribute of the dual. Whenever the dual has to submit a move, it can first get the original strategy to suggest what move should it would have made, and then flip that action.

```
while Game is being played do
    if First Turn then
        create copy of original strategy;
    end
    simulate original strategy;
    update original strategy's history/internal state;
end
return Flip of original strategy's move
```

Algorithm 1: The Dual of a Strategy

4.2 The Joss Ann

A formal definition of the Joss Ann is given in Section 3.1. Given a probability distribution (x, y) the Joss Ann Cooperates with probability x, Defects with probability y, and plays the original move with probability $1 - x - y$. This can be implemented very simply as seen in Algorithm 2. First, a random number between 0 and 1 is generated. The thresholds for Cooperation, Defection and original move are then $x, x + y$ and 1 respectively.

```
while Game is being played do
    p ← Random number;
    if p ≤ Cooperation Threshold then
        Next Move ← C;
    else if p ≤ Defection Threshold then
        Next Move ← D;
    else
        Next Move ← Original choice of Strategy;
    end
end
return Next Move
```

Algorithm 2: The Joss Ann of a Strategy

4.3 Implementation of Fingerprinting

As defined in section a fingerprint function is merely the expected score of a strategy when played against a Joss-Ann transformer of a probe with varying parameters (see definition 4). As part of this project, a numerical implementation has now been included in the Axelrod-Python library. It begins by taking a sample of the x, y values that may define the Joss-Ann Transformer. The strategy then plays a match against a transformer with each of the sampled values. The average score per turn can be calculated at the end of each match which corresponds to the expected score required by the analytical fingerprint function. The whole process can be repeated for reliability and the resulting scores plotted. The player interactions have been modelled as a spatial tournament within Axelrod-Python, where the strategy plays all of the probes and a probe only plays the strategy. For an example with 9 probes, see Figure 4.1.

Whether the numerical fingerprint matches the analytical one relies heavily on the choice of parameters. Specifically the `turns, repetetitons` and `step` variables. The `step` variable determines the number of x, y values taken. Listing 2 shows how a grid of points is constructed over the unit square where the distance between each point is taken as `step`. Therefore, a smaller `step` value means more points are created and so greater detail is included in the plot (similar to pixels).

The `turns` variable determines how many interactions there will be in a match. Enough turns must be selected to ensure that steady long term behaviour is reached otherwise the average score per turn can be wildly innacurate. However, once this state is reached, extending the number of turns has a minimal effect

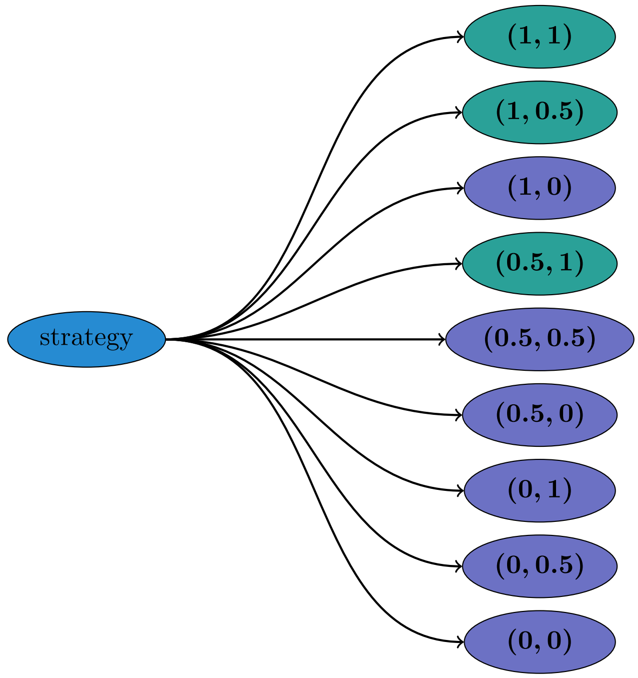


Figure 4.1: A spatial tournament for the strategy against 9 probes

```python
def create_points(step):
    """Creates a set of Points over the unit square.
    A Point has coordinates (x, y). This function constructs points that are
    separated by a step equal to `step`. The points are over the unit
    square which implies that the number created will be (1/`step` + 1)^2.
    Parameters
    ----------
    step : float
        The separation between each Point. Smaller steps will produce more
        Points with coordinates that will be closer together.
    Returns
    ----------
    points : list
        of Point objects with coordinates (x, y)
    """
    num = int((1 / step) // 1) + 1
    points = [Point(j, k) for j in np.linspace(0, 1, num)
              for k in np.linspace(0, 1, num)]

    return points
```

Listing 2: Axelrod-Python code to create a sample of x, y points on the accuracy of the plot. The `repetitions` variable decides how many times the tournament would be repeated. The Axelrod-Python implementation of fingerprinting is a random process (due to the Joss-Ann) and high repetitions helps to reduce the effects of this.

4.4 Comparison of Analytical and Numerical Plots

In figure 4.2, several analytical fingerprints from previous literature are shown [2, 1]. Colourings or shadings are used to make certain features stand out, and an attempt to replicate this behaviour was implemented in Axelrod-Python. The popular plotting library, matplotlib, has many options for different colour maps which are demonstrated in Appendix .

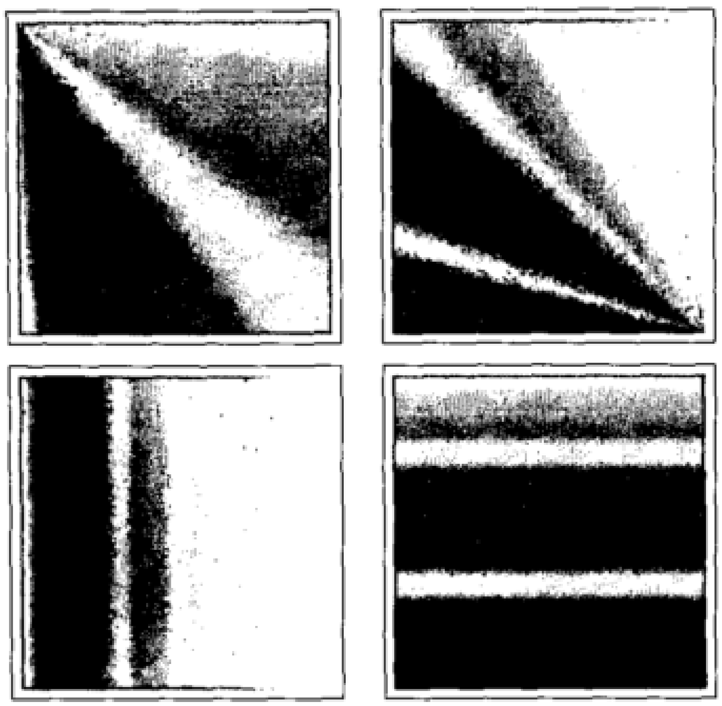
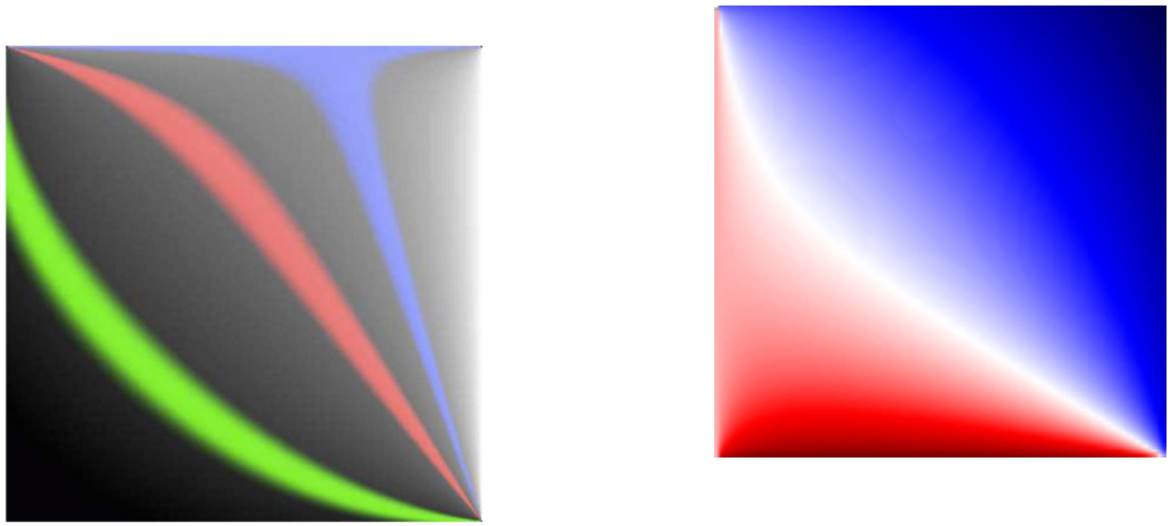


Figure 4.2: Shaded plots of the fingerprint functions for the strategies TitForTat, Psycho, AllD and AllC, in reading order from [2]

Using the analytical fingerprints from previous literature [2, 1], and the fingerprint formulae provided alongside them, the most appropriate colour map was chosen. The colour map Seismic [16] was selected due to its divergent properties (although all colour maps are available within the library). With divergent colour maps, all extreme values (high or low) are coloured, whilst mid range values are left white [25]. This highlights areas of interest, and in Figure 4.3 it can be seen that this matches previous work well.

With the knowledge that the choice of colourmap is appropriate, a comparison can now be made between analytical fingerprints and numerical ones obtained via the Axelrod-Python library. Table 4.1 gives the analytical fingerprint functions of several well known strategies that will then be used to validate the numerical



(a) WSLS fingerprint from previous literature [1]

(b) Analytical WSLS fingerprint demonstrating Seismic colouring

Figure 4.3: A comparison of a fingerprint plot from previous literature to asses the suitability of the Seismic colour map [1]

versions.

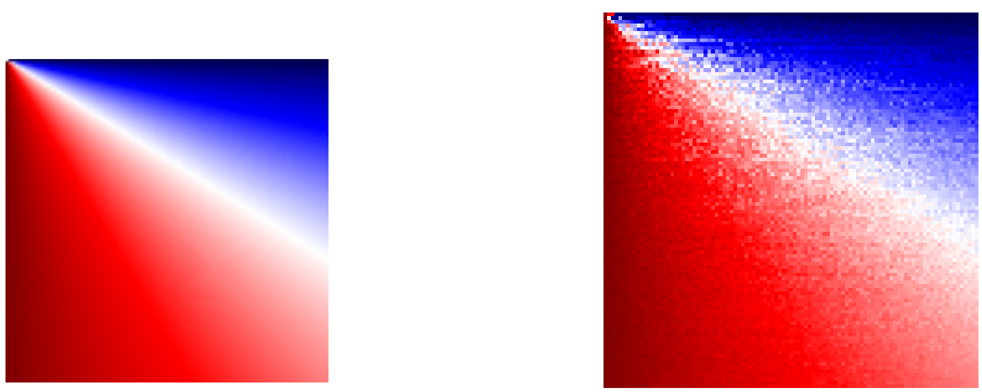
Strategy	Analytical Fingerprint Function
TitForTat	$\dfrac{y^2 + 5xy + 3x^2}{(x+y)^2}$
Psycho (Anti TitForTat	$\dfrac{4(y-1)(x-1) + 5(y-1)^2}{2(y-1)(x-1) + (x-1)^2 + (y-1)^2}$
WinStayLoseShit (Pavlov)	$\dfrac{(3x+y)(x-1) + 5y(y-1)}{(x+2y)(x-1) + y(y-1)}$
AllC (Cooperator)	$3 - 3y$
AllD (Defector)	$4x + 1$

Table 4.1: A selection of analytical fingerprint functions for well known strategies. The probe used is TitForTat.

Figures 4.4 4.5 4.6 4.7 4.8 compare plots of known analytical fingerprint functions with numerical approximations obtained with the Axelrod-Python library. The analytical plots were created with the code seen in listing 3. The parameters `turns=500`, `repetitions=200`, `step=0.01` are as described in section 4.3. The parameter `processes=0` ensures that the function will use the maximum number of cores available on the computer.

```python
import axelrod as axl
strats = [axl.TitForTat, axl.WinStayLoseShift, axl.AntiTitForTat,
          axl.Cooperator, axl.Defector]
for s in strats:
    probe = axl.TitForTat
    af = axl.AshlockFingerprint(s, probe)
    data = af.fingerprint(turns=500, repetitions=200, step=0.01, processes=0)
    p = af.plot()
    p.savefig('{}-Numerical.pdf'.format(s.name))
```

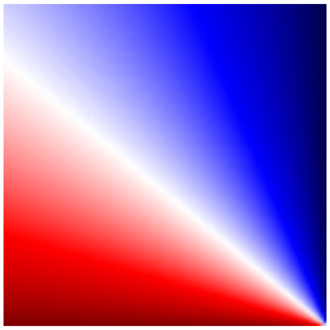
Listing 3: Code to create the numerical plots for several strategies



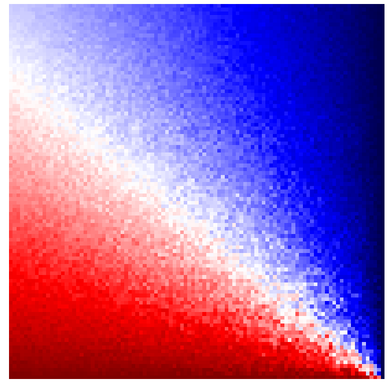
(a) Exact analytical fingerprint

(b) Numerical Fingerprint

Figure 4.4: A comparison of the analytical fingerprint of TitForTat and the numerical version produced by Axelrod-Python library.

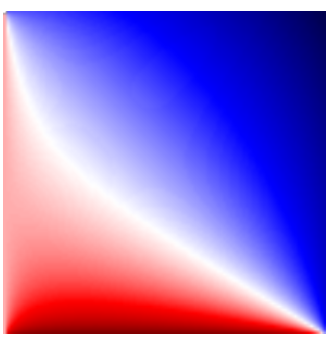


(a) Exact analytical fingerprint

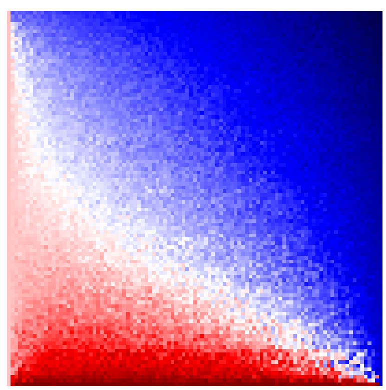


(b) Numerical Fingerprint

Figure 4.5: A comparison of the analytical fingerprint of Psycho and the numerical version produced by Axelrod-Python library.

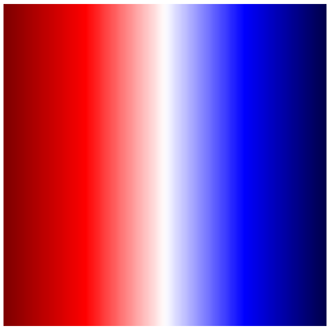


(a) Exact analytical fingerprint

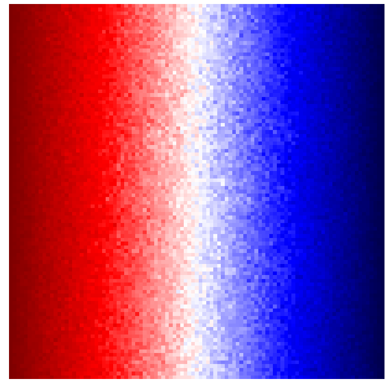


(b) Numerical Fingerprint

Figure 4.6: A comparison of the analytical fingerprint of WinStayLoseShit and the numerical version produced by Axelrod-Python library.

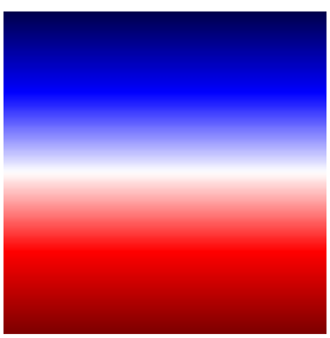


(a) Exact analytical fingerprint

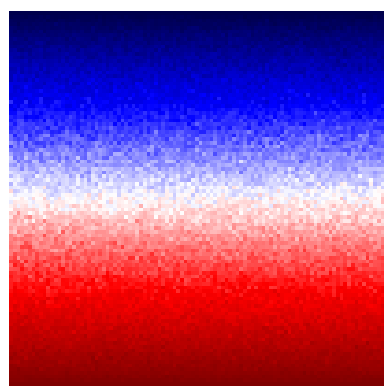


(b) Numerical Fingerprint

Figure 4.7: A comparison of the analytical fingerprint of Cooperator and the numerical version produced by Axelrod-Python library.



(a) Exact analytical fingerprint



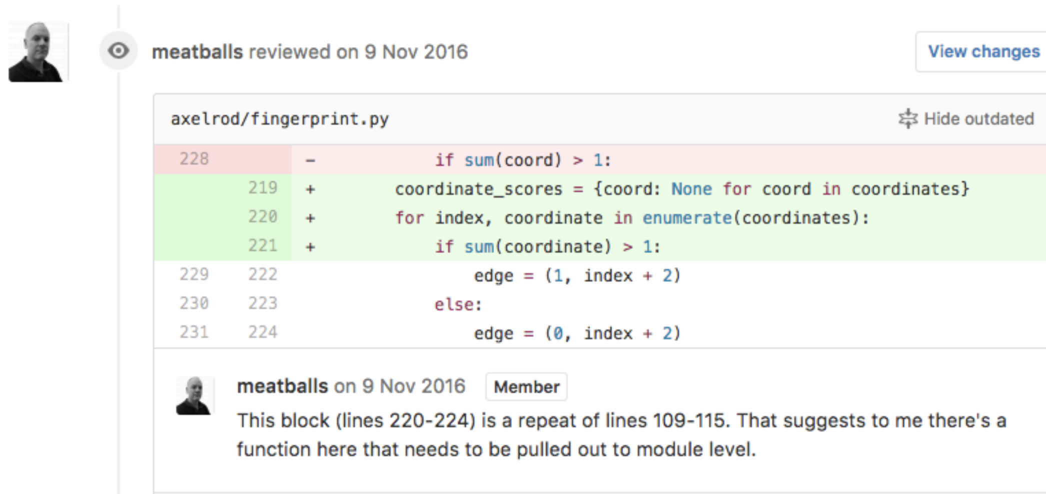
(b) Numerical Fingerprint

Figure 4.8: A comparison of the analytical fingerprint of Defector and the numerical version produced by Axelrod-Python library.

4.5 The Development Process

The Axelrod-Python library aims to follow best practice at all times with regards to development. The source code is hosted at Github and all code is version controlled. Version control ensures that all changes to a code base are tracked and can be traced back to a particular author. It also allow different developers to make sure that they are using the same version of the software, particularly important with regards to reproducing results.

The library also applies a strict review process, where any code submission is analysed by several members of the organisation before it can be included. This normally involves other developers requesting changes to the submission, this ensure that all source code is of the same high standard. Figure 4.5 shows some screen shots of the discussions, requests and suggestions made during the development process of the fingerprint code.



meatballs reviewed on 9 Nov 2016 — View changes

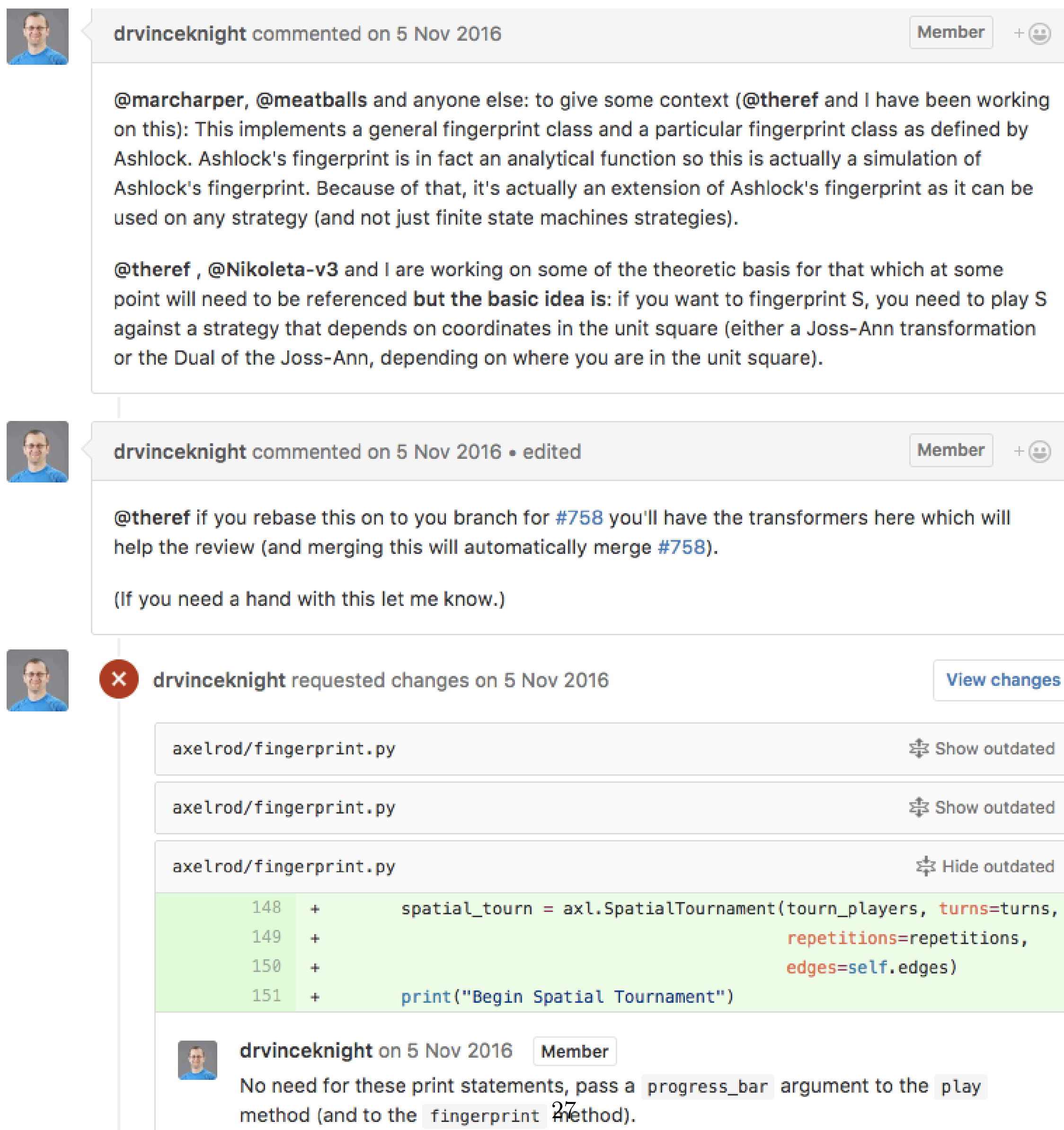
axelrod/fingerprint.py — Hide outdated

```
228      -         if sum(coord) > 1:
     219 +     coordinate_scores = {coord: None for coord in coordinates}
     220 +     for index, coordinate in enumerate(coordinates):
     221 +         if sum(coordinate) > 1:
229  222               edge = (1, index + 2)
230  223           else:
231  224               edge = (0, index + 2)
```

meatballs on 9 Nov 2016 — Member

This block (lines 220-224) is a repeat of lines 109-115. That suggests to me there's a function here that needs to be pulled out to module level.

(c) Exact analytical fingerprint



drvinceknight commented on 5 Nov 2016 — Member

@marcharper, @meatballs and anyone else: to give some context (@theref and I have been working on this): This implements a general fingerprint class and a particular fingerprint class as defined by Ashlock. Ashlock's fingerprint is in fact an analytical function so this is actually a simulation of Ashlock's fingerprint. Because of that, it's actually an extension of Ashlock's fingerprint as it can be used on any strategy (and not just finite state machines strategies).

@theref , @Nikoleta-v3 and I are working on some of the theoretic basis for that which at some point will need to be referenced **but the basic idea is**: if you want to fingerprint S, you need to play S against a strategy that depends on coordinates in the unit square (either a Joss-Ann transformation or the Dual of the Joss-Ann, depending on where you are in the unit square).

drvinceknight commented on 5 Nov 2016 • edited — Member

@theref if you rebase this on to you branch for #758 you'll have the transformers here which will help the review (and merging this will automatically merge #758).

(If you need a hand with this let me know.)

drvinceknight requested changes on 5 Nov 2016 — View changes

axelrod/fingerprint.py — Show outdated

axelrod/fingerprint.py — Show outdated

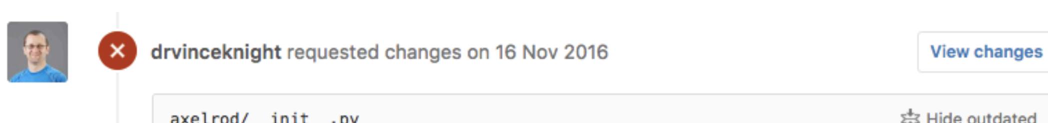
axelrod/fingerprint.py — Hide outdated

```
148 +         spatial_tourn = axl.SpatialTournament(tourn_players, turns=turns,
149 +                                               repetitions=repetitions,
150 +                                               edges=self.edges)
151 +         print("Begin Spatial Tournament")
```

drvinceknight on 5 Nov 2016 — Member

No need for these print statements, pass a `progress_bar` argument to the `play` method (and to the `fingerprint` method).

(d) Exact analytical fingerprint



drvinceknight requested changes on 16 Nov 2016 — View changes

axelrod/__init__.py — Hide outdated

Bibliography

[1] Daniel Ashlock and Eon Youn Kim. "Fingerprinting: Visualization and automatic analysis of prisoner's dilemma strategies". In: *IEEE Transactions on Evolutionary Computation* 12.5 (2008), pp. 647–659. ISSN: 1089778X. DOI: 10.1109/TEVC.2008.920675 (cit. on pp. 10, 11, 21, 22).

[2] Daniel Ashlock, Eun Youn Kim, and Warren Kurt. "Finite Rationality and Interpersonal Complexity in Repeated Games". In: 56.2 (2004), pp. 397–410 (cit. on pp. 10, 11, 13, 21).

[3] Robert Axelrod. "Effective Choice in the Prisoner's Dilemma". In: *The Journal of Conflict Resolution* 65.2 (1980), pp. 217–233 (cit. on pp. 5, 6).

[4] Robert Axelrod. "More Effective choice in the prisone's dilemma". In: *Journal of Conflict Resolution* 24.1 (1980), pp. 3–25. URL: http://jcr.sagepub.com/cgi/content/abstract/24/1/3 (cit. on pp. 5–7).

[5] Robert Axelrod. *The Evolution of Cooperation*. 1984 (cit. on p. 6).

[6] Robert Axelrod and William D. Hamilton. "The Evolution of Cooperation". In: *Science* 211.4489 (1981), pp. 1390–1396. ISSN: 0036-8075. DOI: 10.1126/science.7466396. arXiv: t8jd4qr3m [13960]. URL: http://www.jstor.org/stable/1685895 (cit. on p. 6).

[7] Bruno Beaufils, Jean-Paul Delahaye, and Philippe Mathieu. "Our meeting with gradual, a good strategy for the iterated prisoner's dilemma". In: *Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*. 1997, pp. 202–209. ISBN: 9788578110796. DOI: 10.1017/CBO9781107415324.004. arXiv: arXiv:1011.1669v3 (cit. on p. 7).

[8] J. Bendor, R. M. Kramer, and S. Stout. "When in Doubt ... Cooperation in a Noisy Prisoner's Dilemma". In: *The Journal of Conflict Resolution* 35.4 (1991), pp. 691–719. URL: http://jcr.sagepub.com/content/35/4/691 (cit. on pp. 6, 7).

[9] Siang Y. Chong and Xin Yao. "Behavioral diversity, choices and noise in the iterated prisoner's dilemma". In: *IEEE Transactions on Evolutionary Computation* 9.6 (2005), pp. 540–551. ISSN: 1089778X. DOI: 10.1109/TEVC.2005.856200 (cit. on p. 5).

[10] Siang Yew Chong et al. "Chapter 1 The Iterated Prisoner ' s Dilemma : 20 Years On". In: (2004), pp. 1–21 (cit. on p. 6).

[11] Tom Crick. "Share and Enjoy: Publishing Useful and Usable Scientific Models". In: (2014). arXiv: arXiv:1409.0367v1 (cit. on p. 9).

[12] The Axelrod project developers. *Axelrod: v1.18.1.* Apr. 2016. DOI: 10.5281/168358. URL: http://dx.doi.org/10.5281/zenodo.168358 (cit. on p. 9).

[13] N Franken and a P Engelbrecht. "Particle swarm optimization approaches to coevolve strategies for the iterated prisoner's dilemma". In: *Evolutionary Computation, IEEE Transactions on* 9.6 (2005), pp. 562–579. ISSN: 1089-778X. DOI: 10.1109/TEVC.2005.856202 (cit. on p. 5).

[14] Shaun Hargreaves Heap and Yanis Varoufakis. *Game Theory: A Critical Text.* 2nd ed. 2003. ISBN: 0203199278 (cit. on p. 7).

[15] Neil P Chue Hong et al. "Top Tips to Make Your Research Irreproducible". In: (2015), pp. 5–6. arXiv: arXiv:1504.00062v2 (cit. on p. 9).

[16] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55. URL: http://matplotlib.org/examples/color/colormaps_reference.html (cit. on p. 21).

[17] Hisao Ishibuchi and Naoki Namikawa. "Evolution of iterated prisoner's dilemma game strategies in structured demes under random pairing in game playing". In: *IEEE Transactions on Evolutionary Computation* 9.6 (2005), pp. 552–561. ISSN: 1089778X. DOI: 10.1109/TEVC.2005.856198 (cit. on p. 5).

[18] Martin Jones. *Evolving strategies for an Iterated Prisoner's Dilemma tournament.* 2015. URL: http://mojones.net/evolving-strategies-for-an-iterated-prisoners-dilemma-tournament.html (cit. on p. 8).

[19] Ehud Kalai and William Stanford. "Finite Rationality and Interpersonal Complexity in Repeated Games". In: 679 (1986) (cit. on p. 9).

[20] Ehud Kalai and William Stanford. "Finite Rationality and Interpersonal Complexity in Repeated Games". In: *The Econometric Society* 56.2 (1988), pp. 397–410. URL: http://www.jstor.org/stable/1911078 (cit. on p. 9).

[21] Shyamalendu Kandar. *Introduction to Automata Theory, Formal Languages and Computation.* 1st ed. 2013 (cit. on p. 9).

[22] Vincent Knight et al. "An Open Framework for the Reproducible Study of the Iterated Prisoner's Dilemma". In: *Journal of Open Research Software* 4.1 (2016), e35. DOI: 10.5334/jors.125 (cit. on pp. 7, 9).

[23] David R. MacIver. *Hypothesis 3.6.1.* https://github.com/HypothesisWorks/hypothesis-python. 2016 (cit. on p. 9).

[24] Mojones. *axelrod-evolver.* https://github.com/mojones/axelrod-evolver. 2016 (cit. on p. 8).

[25] Kenneth Moreland. "Diverging color maps for scientific visualization". In: 5876 LNCS.PART 2 (2009), pp. 92–103. ISSN: 03029743. DOI: 10.1007/978-3-642-10520-3_9 (cit. on p. 21).

[26] Tertulien Ndjountche. *Digital Electronics, Volume 3.* 1st ed. Wiley, 2016 (cit. on p. 9).

[27] M Nowak and K Sigmund. "A strategy of win-stay, lose-shift that outperforms tit-for-tat in the Prisoner's Dilemma game." In: *Nature* 364.6432 (1993), pp. 56–58. ISSN: 0028-0836. DOI: 10.1038/364056a0 (cit. on pp. 6, 7).

[28] Martin A Nowak and Robert M May. "Evolutionary games and spatial chaos". In: *Letters to Nature* (1992) (cit. on p. 6).

[29] Martin A Nowak, Robert M May, and Sebastian Bonhoeffer. "Spatial games and the maintenance of cooperation". In: 91.May (1994), pp. 4877–4881 (cit. on p. 6).

[30] James B Procter and Andreas Prlic. “Ten Simple Rules for the Open Development of Scientific Software”. In: 8.12 (2012), pp. 8–10. DOI: 10.1371/journal.pcbi.1002802 (cit. on p. 9).

[31] Rosen and Lawrence. *Open Source Licensing*. 2004, p. 85. ISBN: 0131487876 (cit. on p. 9).

[32] Ariel Rubinstein. “Finite Automata Play the Repeated Prisoner’s Dilemma”. In: *Journal of Economic Theory* 96 (1986), pp. 83–96 (cit. on p. 9).

[33] Geir Kjetil Sandve et al. “Ten Simple Rules for Reproducible Computational”. In: 9.10 (2013), pp. 1–4. DOI: 10.1371/journal.pcbi.1003285 (cit. on p. 9).

[34] Karl Sigmund and Martin A Nowak. “Evolutionary game theory”. In: *Current Biology* 9.14 (1999), pp. 503–505. ISSN: 0960-9822. DOI: 10.1007/978-1-4614-1800-9_63 (cit. on p. 5).

[35] Herbert A. Simon. *Theories of Bounded Rationality*. 1972 (cit. on p. 9).

[36] a. J. Stewart and J. B. Plotkin. “Extortion and cooperation in the Prisoner’s Dilemma”. In: *Proceedings of the National Academy of Sciences* 109.26 (2012), pp. 10134–10135. ISSN: 0027-8424. DOI: 10.1073/pnas.1208087109 (cit. on p. 6).