Generating Solution Sequences for Opponents In The Prisoners Dilemma Using Genetic Improvement Algorithms

Toby Devlin - C1432292
B.Sc. Final Year Dissertation

Cardiff University School of Mathematics

April 3, 2018

Contents

Chapter 1

Introduction

In this report we will be looking best responses of strategies in the Iterated Prisoners Dilemma (IPD); we will try to identify the best overall score, and its corresponding set of moves, for a player on the other side of the game to the strategy. We will look into explicit sequences which should be played for 1 vs 1 games of length 200 to gain the highest score per turn upon the games conclusion.

This task will be completed using reinforcement techniques, specifically genetic algorithms, to continuously improve our sequences during training. Once we have best responses, these sequences will then be compared and contrasted to understand how some opponents could be grouped together even though they have no structure in common.

This work assumes a basic knowledge of game theory and how to model a normal form game. Knowledge if the Prisoners Dilemma is helpful but not required, the IPD game itself is described in Section 1.1 and the real world applications are discussed in Chapter 2.

1.1 The Prisoners Dilemma and Its Iterated Form

The Prisoners Dilemma (PD) is a normal form game in the space $\mathbb{R}^{2\times 2^2}$ with utility matrices and mixed strategies for the row and column player as follows:

$$A = \begin{pmatrix} R & S \\ T & P \end{pmatrix} \quad B = \begin{pmatrix} R & T \\ S & P \end{pmatrix}$$

$$\sigma_r = \begin{pmatrix} C & D \end{pmatrix} \quad \sigma_c = \begin{pmatrix} C & D \end{pmatrix}$$

with the following utility constraints:

$$T > R > P > S, \qquad 2R > T + S$$

These constraints mean that the defection action, D dominates the cooperation action, C, for both player and that the largest payoff comes when both players cooperate. This payoff model has the following interpretation:

- T — Temptation, the utility for successfully tricking your opponent to confess.
- R — Reward, by staying silent with your opponent you receive the reward.
- P — Punishment, if both you and your opponent confess you'll receive the punishment
- S — Sucker, if you confess but your opponent stays silent you're a sucker.

Because of the way the game is put together we arrive at the dilemma. Do you **Cooperate** and risk being taken advantage of by your opponent, first row or column. Or do you **Defect** and hope your opponent stays silent, second row or column.

We will be using the utilities originally from axelrods work,[1].

$$A = \begin{pmatrix} 3 & 0 \\ 5 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 3 & 5 \\ 0 & 1 \end{pmatrix}$$

From a game theoretic point of view it can be seen that there exists a strongly dominant strategy for both the row and column player: defection. This however does not reflect on the iterated version of the game where repetition can allow for higher overall scores and players can take advantage of each other.

We will be looking into the iterated form of the game which, simply put, is just a series of games back to back. Players do not know what their opponent will play on any given turn. Players can algorithmically, stochastically or using a combination of the two decide on their next move. The number of turns is often provided beforehand, but doesn't need to be, and the overall score of the game is normalised to by this to allow for comparisons between games of different length.

1.2 Machine Learning & Computer Intelligence

Machine learning (ML) was most famously introduced by questions posed by Alan Turing in 1950 [2] asking 'can machines think', a question that has been refined and analysed to this day. The field of computer intelligence is rich in its complexities and has recently been making breakthroughs [3] on topics which would traditionally be considered 'thinking'. Along with this, recently there has also been record levels of funding [4] put in to companies which operate in this field, producing results in areas that would usually seem 'solved'.

Computer intelligence breaks down into 2 general topics, supervised and unsupervised learning. Supervised learning is done with help from an external source; inferring a function from labelled examples that will be used to map unseen data to their potential labels. Unsupervised learning is the process of creating new data about a dataset which will be used to expand the understanding of existing data; for example finding a hidden structure or searching a solution space. Reinforcement learning is a technique that can be applied to both supervised and unsupervised learning. What it means is to repeat an algorithm using results from a previous run to improve parameters of its next run; genetic algorithms are a prime example of this.

1.2.1 Genetic Algorithms

This report will cover a class of ML called genetic algorithms (GAs). Genetic Algorithms fall under a branch of ML called evolutionary feature selection algorithms. More generally, genetic algorithms are put into a class of unsupervised reinforcement learning algorithms. These are techniques of using genetic algorithms for generating solutions to problems that typically revolve around heuristically improving members of a population who represent these solutions,[5, 6]. The concept of a genetic algorithm, and more generally an evolutionary algorithm, comes from nature; like nature we create a survival of the fittest selection [7] competition to evaluate a population then kill off the weakest members. After this cull we create offspring from the most successful population or introduce new members from a predefined source. This process is then repeated until we stop it, or forever in the case of nature.

We say a genetic algorithm is structured in the following way. Given a population, P, each with unique genes (genes and member properties are interchangeable), and a number of generations, $G \in \mathbb{N}$, the algorithm will create G cycles of scoring and potentially removing each of the members of the population, $p_i \in P$. It does this by using a mapping from a member of the population to an ordered set, for example $f(p_i) \mapsto \mathbb{R},\ p_i \in P$. This function, f, is defined beforehand in a way which describes the goal of our investigation. Defining a cut-off or bottleneck $b < |P|$, such that on conclusion of completing any cycle, the top b ranking members by score can be kept and the rest discarded. By doing this we are saving the more successful candidates allowing us to rebuild the population using a series of crossovers and mutations (and possibly introducing new members into the population) with the genes which were successful.

- Crossovers take in 2 members of the population and return a new member based on some parameters of the 2 'parents'. For example, our crossover takes the first half of a sequence from one and the second half from the other, merging them to form the third.

- Mutations allow a (possibly targeted[1]) change in a single member of the population. A mutation has 2 parameters, a potency $M_p \in \mathbb{R} > 0$ and a frequency $M_f \in [0, 1]$. M_p describes how strong the mutation is, the higher it is the larger change to the member occurs. M_f explains the percentage of how many members of the population are mutated.

Figure 1.1 shows a flow diagram of a generic genetic algorithm cycle. The specific algorithm we will be using is described in Subsection 3.2.

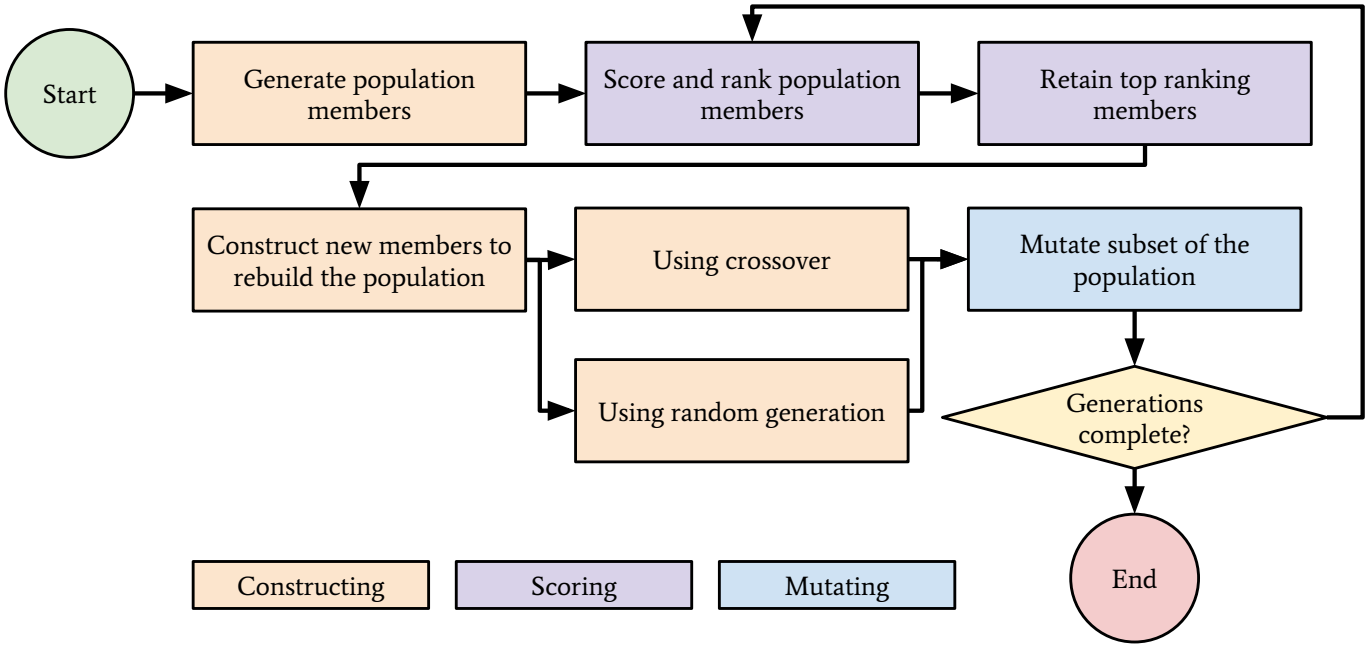

Figure 1.1: Generic genetic algorithm cycle diagram

.

1.3 Brief Task Overview

This work will look at the creation of sequences to beat given opponents in the Iterated Prisoners Dilemma. Analysis will be focused on looking into just the single opponent use case, but the idea of designing a sequence for a tournament for a given number of opponents is a potential follow on to this work. Our task is as follows:

Problem:

When playing a given Iterated Prisoners Dilemma strategy as an opponent, what is the best ordered solution sequence of moves to play in order for us to obtain the highest possible average score per move across the game.

For example an opponent known as Tit For Tat, which cooperates on its first move and the copies your last move on subsequent moves, will have a solution sequence of moves that are all cooperation apart from the last. This is an obvious example and a simple strategy to calculate for, the ultimate goal of this investigation is to look into every strategy as defined in the Python Axelrod Library[8]. These opponents are listed in appendix Section C and definitions can be found in the Axelrod Library codebase.

This problem can be reduced to searching the solution space of every single possible combination of C and D moves. With a game length of 200 however, this is an infeasibly large number of checks and so we will use the genetic algorithm

[1]For example using intuition and targeting specific genes, or allowing another algorithm to improve the targeting of this meta function.

to selectively improve potential sequences. Some examples of 5 move games against Tit For Tat are shown below, the optimal solution is game 5 whereas all the others are sub optimal. This is the process we will be doing for every opponent in the context of a genetic algorithm.

Game 1	1	2	3	4	5	final score	per turn
Tit For Tat:	*C*	*D*	*D*	*D*	*D*	4	0.8
Solution:	*D*	*D*	*D*	*D*	*D*	9	1.8

Game 2	1	2	3	4	5	final score	per turn
Tit For Tat:	*C*	*D*	*C*	*D*	*D*	7	1.4
Solution:	*D*	*C*	*D*	*D*	*D*	12	2.4

Game 3	1	2	3	4	5	final score	per turn
Tit For Tat:	*C*	*D*	*C*	*D*	*C*	10	2.0
Solution:	*D*	*C*	*D*	*C*	*D*	15	3.0

Game 4	1	2	3	4	5	final score	per turn
Tit For Tat:	*C*	*D*	*C*	*C*	*C*	11	2.2
Solution:	*D*	*C*	*C*	*C*	*D*	16	3.2

Game 5	1	2	3	4	5	final score	per turn
Tit For Tat:	*C*	*C*	*C*	*C*	*C*	12	2.4
Solution:	*C*	*C*	*C*	*C*	*D*	17	3.4

Game 6	1	2	3	4	5	final score	per turn
Tit For Tat:	*C*	*C*	*C*	*C*	*C*	15	3.0
Solution:	*C*	*C*	*C*	*C*	*C*	15	3.0

1.4 Structure of this report

This report will contain the following chapters:

- Chapter 1 is this chapter.
- Chapter 2 looks into previous work and literature behind the concepts we will be working with.
- Chapter 3 describes concepts that will be needed to understand results in later chapters.
- Chapter 4 looks into the programming and development that was undertaken to complete the work.
- Chapter 5 looks at how the application of our genetic algorithm was improved before running the final analysis.
- Chapter 6 is the chapter communicating the results of the analysis.
- Chapter 8 describes the takeaways and possible applications of the work.

Chapter 2

Literature Review

In this chapter I will look at previous works in areas relevant to this work. The Prisoners Dilemma was first formally presented by Albert W. Tucker [9, 10] and the iterated version was made famous by Robert Axelrods work in the 1980s [1]. This chapter looks at how these pieces of work have developed and the methods of studying them with computer models.

2.1 Background

The PD and its a large area of repeated games in GT and has applications in the real world. This is due to the game being a good example of strategies that give a cooperative benefit to repeated games. It has been used to describe actions of people and governments in situations stretching from warfare [11, 12]and finance [13] to politics [14] and sexual relationships [15]. Because of its applicability to real world scenarios there is a strong desire to understand how strategies are beaten to either exploit flaws or counter opponents in real world scenarios.

The Prisoners dilemma became a large research area in the combination of mathematics and computer science after Robert Axelrod published his work named 'Effective Choice In The Prisoners Dilemma' [1]. In it he makes an introduction to how tournaments are run and the properties of successful results. His method of experimenting became the standard for handling the IDP problem. This was the first example of computer tournaments, for which he had to as for code programs which had inputs of the history for both players and resulted in an output move for that next turn. After the tournament is complete he describes what successful strategies had in common; It turns out the majority of successful strategies, including the winner, Tit For Tat, have properties of niceness and forgiveness. Niceness is the property of not defecting to start and forgiveness is the property of forgetting previous defections in a timely manner. This allowed them to thrive in the tournament and have overall scores that rose above strategies without niceness or forgiveness.

Since Axelrods' original tournament there have been many research papers on what makes a successful strategy for a specific objective. For example William H Press and Freeman J Dyson, [16], looked at how to it is possible to deterministically set an opponents score and Shashi Mittal and Kalyanmoy Deb have looked at a range of different objectives at once [17]. These specific objectives are the core part of applying Game Theory and the Prisoners Dilemma in the real word [18, 19, 20]. Objectives are the wrapper for which we can work with real world scenarios, for example in a the cold war it was not the goal to get as many missiles as possible to pass an oppositions defence but to not allow any missiles through your own defence; i.e.minimising your opponents score. In a football tournament where winning is the number or goals your team scores it doesn't really matter how many goals you get in so long as you score the most overall. This leads to observing that the IPD describes that a players world is better if their opponent is cooperative. There are also some very useful applications such as rent splitting or work assignments [21], all of which were programmed and deployed to a website for general use online [22].

2.2 Strategy Structures

A strategy is a way of defining how to play the IPD. It is a way of identifying which move to play next based on some parameters; for example the strategy of all C moves is valid, as is a wildly complicated description of what to do based on the last 20 moves of an opponent.

Individual strategies can be defined in multiple ways [23]. Each method of representing a strategies has its benefits and drawbacks, typically due to fundamental properties of the strategy (like being stochastic) or that it is more concise to write in a certain method. There are methods of creating strategies that remove the way opponents play in order to create overall scores that desired. Ways of structuring a strategy are shown below:

- LookerUpper, for example figure 2.2.
- Gambler, for example the Stochastic stratergies; examples can be found in [16].
- Neural Networks.
- Finite State Machines, for example figure 2.1.
- Hidden Markov Models.
- Explicit Move List, for example the solutions given in Appendix D.
- Mixtures.

2.2.1 Equivalent Strategies

Looking at certain opponents there are occasions some strategies that look indistinguishable from others; for example two stochastic opponents can can be incredibly similar, but identifying which one is which from their play history could be impossible. One of the ways we are able to identify strategies is the process of fingerprinting [24, 25, 9]. However this can be inconclusive and less accurate as desired. An effective method of identifying equivalent strategies is to write them down in the form of the other. For example if we can write any strategy in one of the forms given in figures 2.1,2.2 we know its an equivalent to Tit for Tat.

The work being completed in this paper is another method of identifying strategies; identifying the best response to them may lead to observations about how and why different strategies act in similar manners. This wont lead to show strategy equivalence, but it will show how solution equivalence.

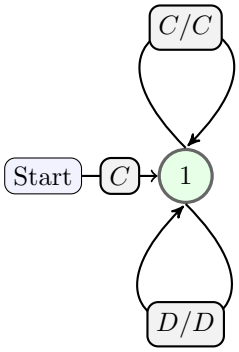

Figure 2.1: Finite State diagram of strategy Tit for Tat

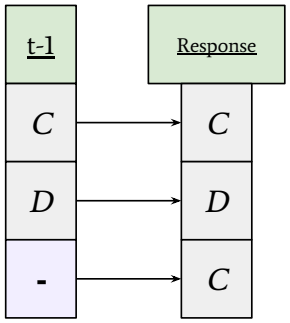

Figure 2.2: Look Up diagram of strategy Tit for Tat

2.3 Strategies Of Interest

Tit For Tat was the winner of the original axelrod tournament in 1980 [1]. It s a very basic opponent who has strong forgiveness (it will forgive a defection after 1 move) and generosity (it will start with a cooperation) which is thought to give its well overall score in tournaments.

Alternator is a 'dumb' opponent, i.e. no strategic method at all. All it will do is alternate between cooperation and defection until the game ends. Playing an Alternator effectively is to defect the whole game, however identifying an alternator to play this sequence without backlash can be difficult if we're playing a potentially similar strategy such as Random or

Grudger can be considered as the most unforgiving strategy that exists. Starting by cooperating, if you defect even once then the Grudger will defect until the end of the game. In line with Tit For Tat, our best score will come from not 'upsetting' the opponent (until the last move where it cant react.

Random is the most basic stochastic opponent, and like Alternator and Cycler it is 'dumb'. With a probability p of cooperating and $1 - p$ of defecting, we can just defect the whole time to beat this opponent; picking up bonus points on its cooperation moves.

EvolvedFSM16

CollectiveStrategy has the property of a handshake? it will try to identify if the opponent is another

ZDExtortion This is the paper [16]

Cycler

2.4 Genetic Algorithms

This work will focus on GAs which are a specific form of ML. Advanced techniques of machine learning can be combined and used together[1] in many situations, lots of these techniques combine genetic algorithms or some sort of fitness testing within a larger scope. The field of mathematics research is one which has plenty of examples of GAs in action; for example the same techniques as we will using is used in [27] to solve the Generalised Assignment Problem. Another example, [28], used neural networks when approaching the state regulation problem.

[1] Techniques for teaching and versioning static algorithms such as building a 'clever' game AIs, where the core concept of the AI is fine tuned using GA in an development environment but isn't implemented into the game [26].

Chapter 3

Task Background

This chapter will cover sections on how each part of the project will been carried out. Useful notation is given in Section 3.1 which will allow us to quickly describe a given sequence or set of sequences in a compact form. We will also look at our specific instance of genetic algorithm that exists in the Axelrod Dojo codebase. Lastly Section 3.3 looks at the what results we will expect from the analysis and the potential problems we want to mitigate.

3.1 Notation

Let $S \in \{C, D\}^L$ where C, D represents a cooperation, defection respectively. $L = 200$ is used throughout this report. We can split up sequences into blocks of consecutive move elements of the same type. We will use B_i to denote the block after i changes of move type from the explicitly stated starting move type.

- Every move in a block is of the same type; the type is implicit based on whether i is even or not and what the starting move type was.
- We can use the notation $|B_i|$ to denote the length of the ith block in the sequence.$|B_i| \in \mathbb{Z}$

This means we can write a sequence as a series of blocks:

$$S = B_1 B_2, \ldots, B_n$$

A Sequence can also be defined shorthand by specifying the starting move and the length of subsequent blocks:

$$S = C\ |B_1|, |B_2|, \ldots, |B_n| \quad \Rightarrow \quad S = \overbrace{C \ldots C}^{|B_1|}\overbrace{D \ldots D}^{|B_2|} \ldots \overbrace{(C|D) \ldots (C|D)}^{|B_n|}$$

We can also construct sequences from repetitions of a sequence of blocks when it makes sense:

$$C\ \left(|B_1|, |B_2|,\ \ldots,\ |B_m|\right)^k \quad \Rightarrow \quad \overbrace{C \ldots C}^{|B_1|}\overbrace{D \ldots D}^{|B_2|} \ldots \overbrace{D \ldots D}^{|B_m|}{}^{k-times}$$

The two notations can be combined to add starting and ending blocks to a repeating sequence (shown in the examples).

It is also possible to define sets of sequences by adding variables to parameters of the sequence. Appropriate selection of parameters mean the length of sequences shouldn't grow.

$$\{Ci, l-i\} \quad i \in [a, b] \Rightarrow \{\underbrace{C \ldots C}_{a}\overbrace{D \ldots D}^{l-a},\ \underbrace{C \ldots C}_{a+1}\overbrace{D \ldots D}^{l-(a+1)}, \ldots,\ \underbrace{C \ldots C}_{b}\overbrace{D \ldots D}^{l-b}\}$$

For long sequences where there is no recognisable pattern it is typically easier to describe the solution. Otherwise we use the notation to describe a sequence. When we look at solutions there too long to write the sequence directly, in which case we may abbreviate in the following style: $C1, 5, 2, 3, 5, 6, ... = CDDDDDCCDDDCCCCCDDDDDD....$ Examples:

$$C : [1, 4, 3, 2] = CDDDDCCCDD \tag{3.1}$$

$$D : [\ (1, 1)^5] = DCDCDCDCDC \tag{3.2}$$

$$C : [1, (2, 1)^2, 2, 1] = CDDCDDCDDC \tag{3.3}$$

$$\{D : [i, 5 - i]\} \quad i \in [2, 4] = \{DDCCC,\ DDDCC,\ DDDCC\} \tag{3.4}$$

$$\tag{3.5}$$

3.2 Building The Algorithm

In order to generate a solution sequence we have to train against each Axelrod opponent against another Axelrod opponent. We will use the Cycler strategy[1] in the algorithm by editing the input parameter to improve our score against an opponent.

Our implementation of a genetic algorithm is more custom and has the following steps:

1. Start with a predefined population, supplemented with randomly generated member until to size.
2. Each member plays the given opponent with their sequence and returns with the average score per turn.
3. Members of the population are ranked by this average score per turn and the highest scoring 25% will be kept for the next round. The remaining 75% are killed off.
4. The remaining population will then be copied and these copies mutated to create unique sequences before being merged back in to the main population.
5. The remaining 50% difference is then made up of mutated results of crossovers from members of the current population or random new members, depending on a random selection algorithm.[2]
6. A generation has now concluded. Repeat from step 2 until the desired number of generations are finished and a final best sequence is returned.

Figure 3.1 shows a flow diagram of our cycle. This is the algorithm we will use in Chapter 5 when analysing the algorithms parameters The model means we can create a population of Cycler players and input a sequence of length 200 as a parameter to set off our genetic algorithm. The subsequent inputs for the populations Cycler players will be created using the genetic mutation and crossover techniques, see Section 1.2.1 for details.

The looping will be the basis of creating the optimal strategy for each other opponent. Each step is defined in the Axelrod Dojo `Population` and `CyclerParams` classes. Rather than store all the functionality in one place we are able split up aspects of the flow to allow for flexibility in what type of population can be used.

3.3 Solution Form

In this research our goal will be to use the algorithm, described in Subsection 3.2, to produce an arbitrary sequence for an opponent, S_o. This sequence will represent the moves we should play against the opponent to get our largest potential score per turn for a single game of 200 turns.

This investigation focuses on sequences that will allow us to maximise our score overall, rather than just beating any given opponent. An analogy of this concept is a team playing a football tournament, but instead of a knockout competition

[1] See Section **??** for description

[2] This algorithm had a bug which would change the size of the population in the first generation. This was fixed after Section 5.7 was written. See: https://github.com/Axelrod-Python/axelrod-dojo/issues/43

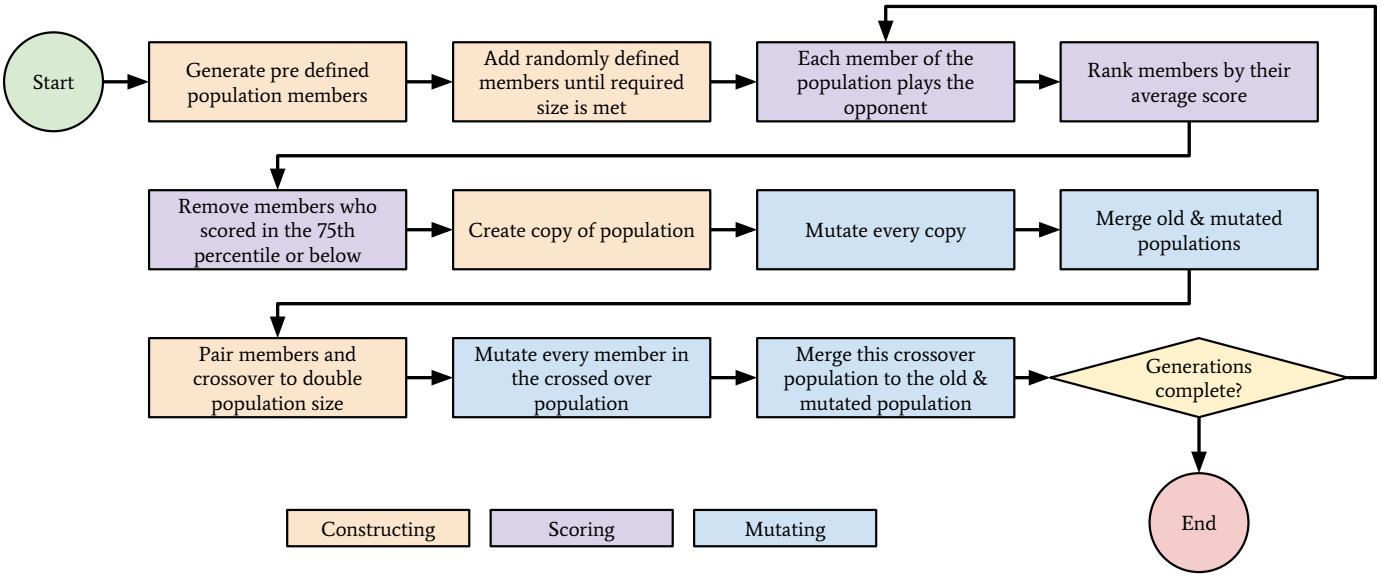

Figure 3.1: High Level Genetic Algorithm Cycle. Extension of Figure 1.1

our team is placed in the standings based off the total goals they have scored across the tournament. More real world applications of these results are discussed in Chapter 7.

Each solution sequence is uniquely generated for each opponent. If there are two similar opponents, say Grudger and Collective Strategy, these will be independently analysed and individual solutions will be generated. The output sequences will be compared on how similar they are to each other, what type of opponents and their corresponding 'best scores' are and whether there are patterns to how to group strategies, Chapter 6 contains further details.

When working with stochastic players, we will be seeding them in order to determine the best sequence. Each stochastic player will be considered under a variety of seeds. The motive to allow this to happen is explained described in Section 5.6.Non-stochastic opponents will be seeded in the code, however there is no change in their behaviour because of this.

3.4 Conclusion

Here we have looked at the background required to answer a specific question, build a relevant model and understand what were looking for. The concepts themselves are extensions to the basic models brought up in Chapter 1 and should allow us to provide relevant results in a uniform manner.

We have also looked at what problems may arise from certain areas of development, such as understanding what a best response for a stochastic opponents could be or how to represent a complex solution in a result. Because of this Chapters 5 and 6 will be concise.

Chapter 4

Developing The Codebase

This chapter will discuss the development practices encountered during the project and a walkthrough of setting up an environment. Details on the set up and reasoning behind using specific development environments will be given along with a look at the open source contributions made. A brief look at version control will also show how code is developed to be sustainable and reproducible.

4.1 Research Environments

A mix of Jupyter Notebooks and integrated development environments[1] were used to write and execute code. The main analysis was run using a factory class pattern, called `AnalysisRun` in the `full_analysis` module, show in Figure B.2. This class was used to wrap a query to the Axelrod-Dojo functionality and subsequently to the Axelrod Library in such a way that was easy to control batch executions.

The analysis itself was done using native multi-threading on a Linux OS to improve individual opponent analysis run times and the overall scalability of the project. .This tutorial will also assume you're working with a local development station; analysis on remote cloud instances of Jupyter Notebooks is possible but the set up is different.

Installing Basic Libraries Your first step should be to download and install the Anaconda distribution for your OS here: https://www.anaconda.com/download/. This will allow you to use the integrated c++ libraries python has to offer without needing to mess about too much. Anaconda also has them majority of Functional Libraries above and Jupyter Notebooks pre installed to make setting up much easier. From here, follow the instructions the install wizard has to add any environment variables to allow CMD/Bash access to binaries.

Installing the Axelrod and Axelrod-Dojo libraries uses the pip tool that already comes with Anaconda and should be ready to execute after the last step.Running 'pip install axelrod' then 'pip install axelrod-dojo' will install these.

Once this is installed the `full\_analysis.py` file has to be downloaded from github[2], it can be found in the code directory.This can just be copied and pasted if needed all were interested in is the class to generate a sequence for an opponent.

Running a Test Figure 4.1 is some sample code that will run an analysis with the following settings:

- Override the default mutation frequency of 0.1 to 0.3.

[1] Pycharm Professional Edition and Microsoft VS Code.
[2] https://github.com/GitToby/FinalYearProject

- Set the prefix for all the files to be 'example-'.
- Analysing 3 opponents.(Random will have multiple instances for different seeds.)

```
from full_analysis import NewAnalysisRun
import axelrod as axl

run = NewAnalysisRun(mutation_frequency=0.33)
run.save_file_prefix = "example-"

run.add_opponent(axl.TitForTat())
run.add_opponent(axl.Random())
run.add_opponent(axl.Grudger())

run.start()
```

Figure 4.1: Code to create a sequence result to optimise best score for 3 opponents

After it has run the generated data output should be stored in the './output' directory. If the code fails to run there may be issues with this directory being created. There should be multiple output files, each with one opponents evolution stages through the generations.

Cloud Notebook Setup If you want to use a Cloud service, such as Azure Notebooks or AWS Sagemaker, the set up is similar to the above just executed differently. Installing Anaconda is not needed, the environment has the required installs. Using pip to install the Axelrod libraries and download the full analysis script can be done in an integrated web terminal or directly in a notebook. Figure 4.2 shows and example in azure, copy these in your top few cells of your jupyter notebook and it will work as required.

```
# -------- CELL 1 --------
# The ! means 'run this as a bash script'
! pip install axelrod
! pip install git+https://github.com/Axelrod-Python/axelrod-dojo.git
! wget https://raw.githubusercontent.com/GitToby/FinalYearProject/master/code/full_analysis.py

# -------- CELL 2 --------
import axelrod as axl
import full_analysis as fa

run = fa.NewAnalysisRun(population_size=40)

run.add_opponent(axl.TitForTat())
run.add_opponent(axl.Random())
run.add_opponent(axl.Grudger())

run.start()
```

Figure 4.2: Cells for creating the jupyter instance of a research environment

4.2 Codebase Contributions

Throughout the project I had split time between writing my own code and expanding the Axelrod Dojo codebase. In modern software development there is a commitment in the developer community to track and reuse as much code as

possible, typically using a version control system such as Git or Subversion. The majority of open source community development is conducted on github[29] using Git, the Axelrod libraries, along with most other libraries I used, are hosted here.

When writing code in a professional environment there is a predetermined scope that all parties agree upon before the work commences. This, however, is not the case for research development which is more organic; final products that are created when conducting research for papers are highly personalised and typically cannot be used in other areas. This leads to more flexible products that operate more as platforms or tools for further research, this is why the Axelrod, Axelrod Dojo and other libraries mentioned in table 4.1 exist. My final code will only be used for researching the IPD in my projects specific direction, this meant extending the platforms to handle what I needed them to do, subsection 4.2.1 looks into how this was completed in my project.

4.2.1 Version Control

During the project I created a 'fork' of the axelrod dojo in order to add content to the open source community. This resulted in using Git to create a new 'branch' on which to write my code before asking the owners of the repository to pull my work back into the core product using a 'pull request'(PR). The PR opened for my code[3] resulted in changes to 457 lines of code and 14 files, adding classes and fixing bug that had been previously flagged. Figure 4.3 shows the initial scope of the PR and Figure 4.4 shows a section of the commits made before merging the branch.

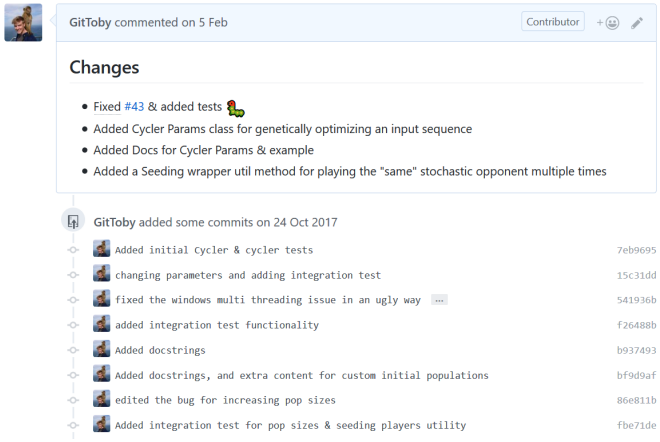

Figure 4.3: Description and commits for PR on Github

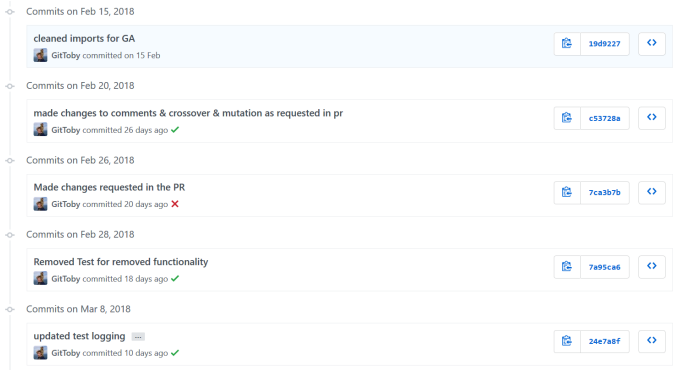

Figure 4.4: Tail of code commit log as shown on Github

After a request is opened there is a period of reviews by the owners, this is to ensure code quality and scope coverage. As my work progressed I continued to add to the PR which lead to more and more requests being added for features that fell outside the scope of the PR. Eventually we decided to create another fork of the PR that contained specialist code for my project that wouldn't benefit the codebase as a whole. Figures 4.5 & 4.6 show examples of requests and discussions around features and code quality. Because of this review process the overall quality of the codebase can be kept high and the owners can decide on how their platforms are developed.

4.2.2 Testing

Code testing and version control go hand in hand. When new releases of code are made it is important to ensure that the new changes dont break previous functionality. This is kept in check by the presence of continues integrations (CI) and test environments. As of this project the CI for the Axelrod Dojo keeps the libraries tests running on a linux environment and the output fed to the Github page. During my development I had to implement tests for any functionality I wrote to ensure the library still worked as intended. Following the mixed practice of test driven design (TDD) and behaviour driven design (BDD) the code written to extend the Axelrod Dojo had tests to cover examples of what happens in a production environment. Snippet 4.7 shows an example of a test.

[3] https://github.com/Axelrod-Python/axelrod-dojo/pull/45

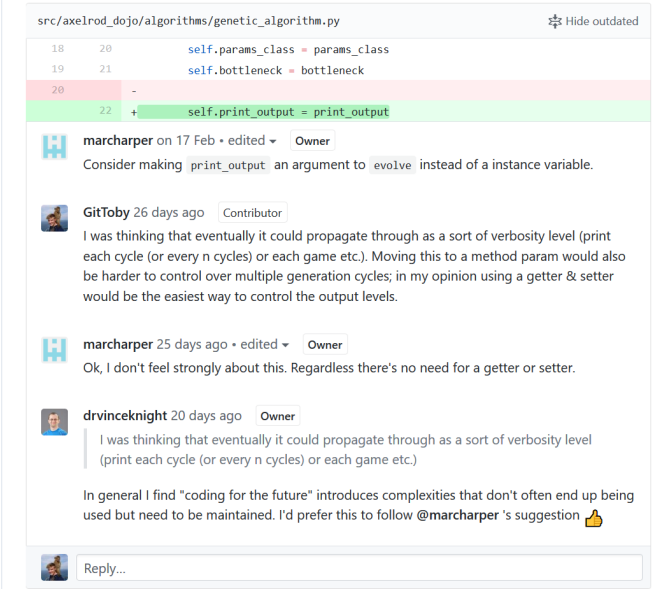

Figure 4.5: Feature discussions in PR on Github

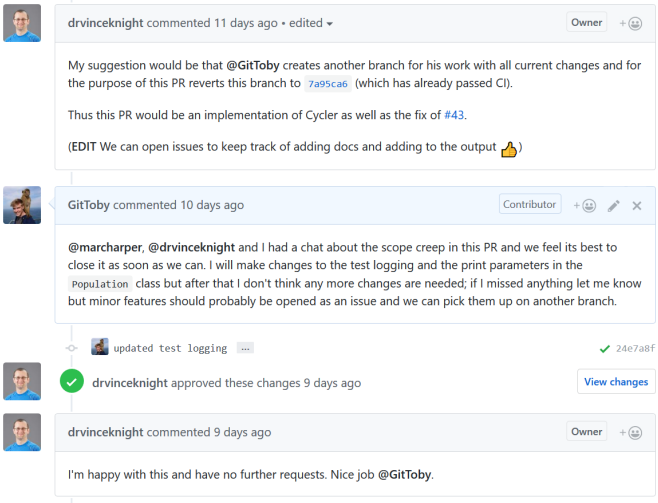

Figure 4.6: Scope Discussion in PR on Github

```python
def test_creation_seqLen(self):
    axl.seed(0)
    test_length = 10
    self.instance = CyclerParams(sequence_length=test_length)
    self.assertEqual(self.instance.sequence, [D, C, C, D, C, C, C, C, C, C])
    self.assertEqual(self.instance.sequence_length, test_length)
    self.assertEqual(len(self.instance.sequence), test_length)
```

Figure 4.7: An Example of a test in the Axelrod Dojo

4.3 Libraries And External Modules

Main Research Libraries

- **Axelrod** — Used for the core of the prisoners dilemma and iterated prisoners dilemma functionality code.[8]
- **Axelrod-Dojo** — Applied machine learning techniques that revolve around generating solutions to questions relating to the Axelrod library.

Functional libraries Table 4.1 shows the external functional libraries used, while Table 4.2 shows the internal python built in modules that where leveraged during development. These are libraries which are not involved in the core functionality of the IPD.

Library	Reason
matplotlib pyplot	For plotting graphs and images with data
pandas	For data manipulation.
numpy	For reducing complexity of numerical calculations.

Table 4.1: Functional Python libraries for analysis

Module	Reason
os	For operating system functionality.
time	For time calculations.
itertools	For easier iterations over data structures.

Table 4.2: Internal built in python modules used

4.4 Conclusion

In this chapter we covered the code and techniques needed to complete research. Of the learning that was complexity during the project, learning a new version control system was the hardest along with handling the organic growth of the project. While writing code there where sections that needed changing or removing, however there was no scope or predetermined set of outcomes causing my own analysis notebooks to swell and become unwieldy at times. These problems were solved by selectively isolating work and levering 3rd party libraries more.

The final codebase contributions were a success and version 0.0.8 of the Axelrod Dojo has been released to the python installer package, 'pip'.[30] Following the steps in Section 4.1 will allow a working environment to be set up on any platform, letting this experiment be repeated. Providing a reproducible experiment allows the open source and research communities independently verify any findings in this work to ensure a robust foundation for future work. All finalised notebooks and final code are published online on Github, urls can be found in Appendix A.

Chapter 5

Implementation Of Sequence Discovery

In this chapter, consideration will be given to the process of finding the optimal sequence of moves against another player. The various approaches used and a detailed analysis of the optimisation procedures and parameters will be described.

Before conducting the bulk calculations for the set of opponents listed in the appendix we will test values for the algorithms' parameters to see what the best settings are for finding solution sequences. We will describe a series of sequences as 'converged' if the best score over the set number of generations has reached a stable point; described as when none of the moves in the sequences has changed over a number of generations. This stable point, however, may not be the optimal solution. we may have found a local maxima for the solution sequence rather than the global maximum.

The opponents we select for this section are in some way interesting. They are mostly 'simple' and can be explained in a sentence or two, but each one has a fundamentally different structure to how they work. We will look into these as we can confirm that the genetic algorithm will select the optimal sequence solution for the selected type of opponent.

What we want to look at is how the best score rises over generations as we change certain features of the algorithm. Once the best score per turn hits a maximum such that it wont change no mater how many more generations are run, as described in Section 3.3, this the optimal solution sequence. for a solution to be unique we must be playing a non stochastic opponent.

Once a solution has been found the generation number where this plateau occurs is called the solution sequence distance, or solution distance; one of the goals of this initial investigation is to see how parameters affect the distance. During the investigation we may find solutions that are not optimal, meaning that the algorithm will have found a sequence that will do well against an opponent but wont find the best sequence that will return can possibly get. These sub optimal solutions are due to the occurrence of local maxima in the set of scores of neighbouring sequences. Genetic algorithms are designed in way to avoid local maxima's; the property of mutating (i.e jumps of their features) allow members of the population to potentially remove themselves from these local maximas. We will look in depth into how to overcome the possibility of our algorithm finding a local, rather than, global maximum in Section 5.4. Some of the questions we will hope to be answering include:

- If we have a larger initial population sample to start with, will we reach our maximum best score earlier?

Player	Optimal Sequence	Representation
axl.TitForTat()	$CCC \dots CD$	$Cn-1,1$
axl.Alternator()	$DDD \dots DD$	Dn
axl.Grudger()	$CCC \dots CD$	$Cn-1,1$
axl.Random()	$DDD \dots DD$	Dn
axl.EvolvedFSM16()	$CDC \dots DD$	$C1,1,n-2$
axl.CollectiveStrategy()	$CDC \dots CD$	$C1,1,n-3,1$
axl.Champion()	Various[1]	NA

Best Score	Gen	Mean Score	Population	Sequence	Std Dev	Time Taken
2.425	1	2.264	**25.0**	DD...	0.067	6.646
2.425	2	2.343	**25.0**	DD...	0.046	6.646
2.425	3	2.393	**25.0**	DD...	0.038	6.646
...	...	...	...	...	...	...
2.830	102	2.782	**100.0**	CC...	0.112	28.425
...	...	...	...	...	...	...
2.980	150	2.911	**500.0**	CC...	0.158	152.684
...	...	...	...	...	...	...

Table 5.1: Output data table

- What about increasing the generations, is there an optimal number of generations to run the algorithm for such that we always find a solution sequence.
- If we make each sequence more likely to mutate generation to generation what will happen? What about increasing how potent our mutations are?

5.1 Changing Initial Population Size

The initial population size is the number of starting sequences we use in our algorithms first generation. Once this generation concludes the population will go through the series of phases outlined in figure 3.1; altering the population to keep the best performers against our opponent to continue on to subsequent generations. During any given generation the population defines the maximum potential range of scores that we can achieve against our opponent. For example, having 2 members with distinct sequences in our population would provide us with 2 distinct new sequences after mutation. Because of this we can reasonably assume the larger our population the larger the number of distinct scores leading to a larger chance of finding the solution sequence with the optimal score; hence we should converge to the solution sequence in less generations.

The implementation of analysing a range of populations requires us to understand how the solution distance is affected as we run our algorithm through a set of population sizes, say $|P| \in [25, 50, 100, 150, 200, 250, 500]$.

EFFICIENCY NOTE: Increasing the size of our population will have an impact on computation time; each generation must process the full population in a linear fashion causing a computation overhead of $O(n)$. For an increase to be useful a in any time restricted scenario our algorithm would need to show a higher order benefit in our distance to convergence, or in our average score per turn. However We are not working in a time restricted scenario, and so we should just select the best overall initial population size independent of computation overhead. In a perfect world where everything was time independent we would brute force every possible sequence combination

The code in Snippet 5.1 is an implementation how we go about analysing and storing the tests on generation sizes listed. It leverages the use of the function `runGeneticAlgo` show in appendix Snippet B.1. This code will output data in the form of Table 5.1.

By grouping this data by the population we observe how initial populations affect different opponents. Its clear that from figure 5.2 that the initial population size has a significant effect on finding better sequences. We can see if there is a larger initial population there is typically a higher best score shown once concluding all of the generations. It doesn't, however, ensure that we find the solution sequence; as is shown in the lack of long plateaus in the lines.

The improvement's from this effect are non-linear from observation. The change in final best score for a population of 50 compared with a population of 100 is large in comparison to the same relative increase from 200 to 250. This may suggest there are more efficient approaches to improving our final best score after a certain initial population is reached.

None of these results have found a solution sequence (or at least we cant tell from the graph). It is clear that larger initial populations do, on a relative scale, much better than small ones. There are no large plateaus for the graph, so as we

```
def populationChecker(opponent):
    # make a nice file name
    file_name = "data/" + str(opponent) \
        .replace(" ", "_") \
        .replace(":", "_") \
        .lower() + "_pop.csv"

    # if the file exists don't run_one, it takes forever, make sure it exists
    if not os.path.isfile(file_name):
        df_main = pd.DataFrame(data=None, columns=col_names)
        for pop_size in populations:
            start_time = time.clock()
            pop_run = runGeneticAlgo(opponent,
                                     population_size=pop_size,
                                     number_of_game_turns=200,
                                     cycle_length=200,
                                     generations=150,
                                     mutation_probability=0.1,
                                     reset_file=True)
            end_time = time.clock()
            tmp_df = pd.read_csv(pop_run[0], names=col_names)
            tmp_df["population"] = pop_size
            tmp_df["time_taken"] = end_time - start_time
            df_main = df_main.append(tmp_df, ignore_index=True)
        df_main.to_csv(file_name)
        print("List Complete:", file_name)
        return df_main
    else:
        print("file already exists, no calcs to do.")
        file_df = pd.read_csv(file_name)
        # remove first column
        file_df = file_df[list(file_df)[1:]]
        return file_df
```

Figure 5.1: code to check multiple populations

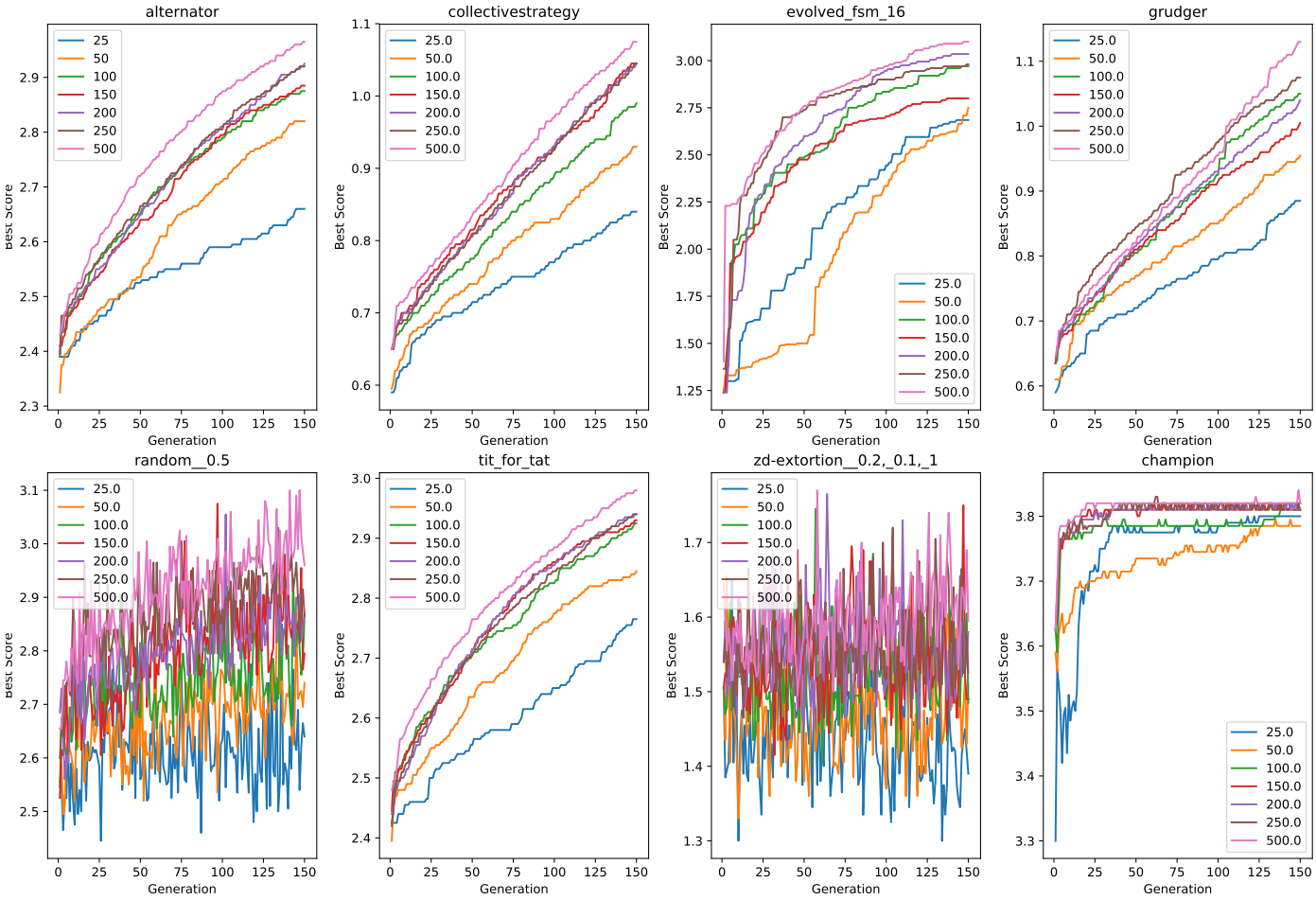

Figure 5.2: Best score per turn vs generation for different initial population sizes

continue our research the initial population size will be increased to 150 to keep computation times manageable.

5.2 Generation Length Analysis

Another major component parameter of a genetic algorithm is the number of generations it will run before outputting a final sequence. The number of generations has an influence on a number of different things within the algorithm:

- The total combinations of features, (sequence elements), that the algorithm can test.
- The number of low performers we can remove in our population.

For our goal of finding the optimal solution sequence for each opponent it would be useful to extend the generations as far as possible; this would provide the most combinations of features possible. Here we will look into how close to a solution sequence we get when we increase the generations the algorithm runs for. Like in previous experiments with other variables we will use a set of sizes for our parameter to run our analysis for each size, here we will take generation lengths $G \in [50, 150, 250, 350, 450, 500]$.[2] The code in Snippet 5.4 shows how we will approach the analysis.

Generation size differs from other parameters in the fact this is purely performance based. A genetic algorithm with 1 generation is just a series of tests; with the results split into 2 sets — winners and losers. As we extend the generations

[2]We will be using a population of 150 as this was the best average for score vs computation time for analysis.

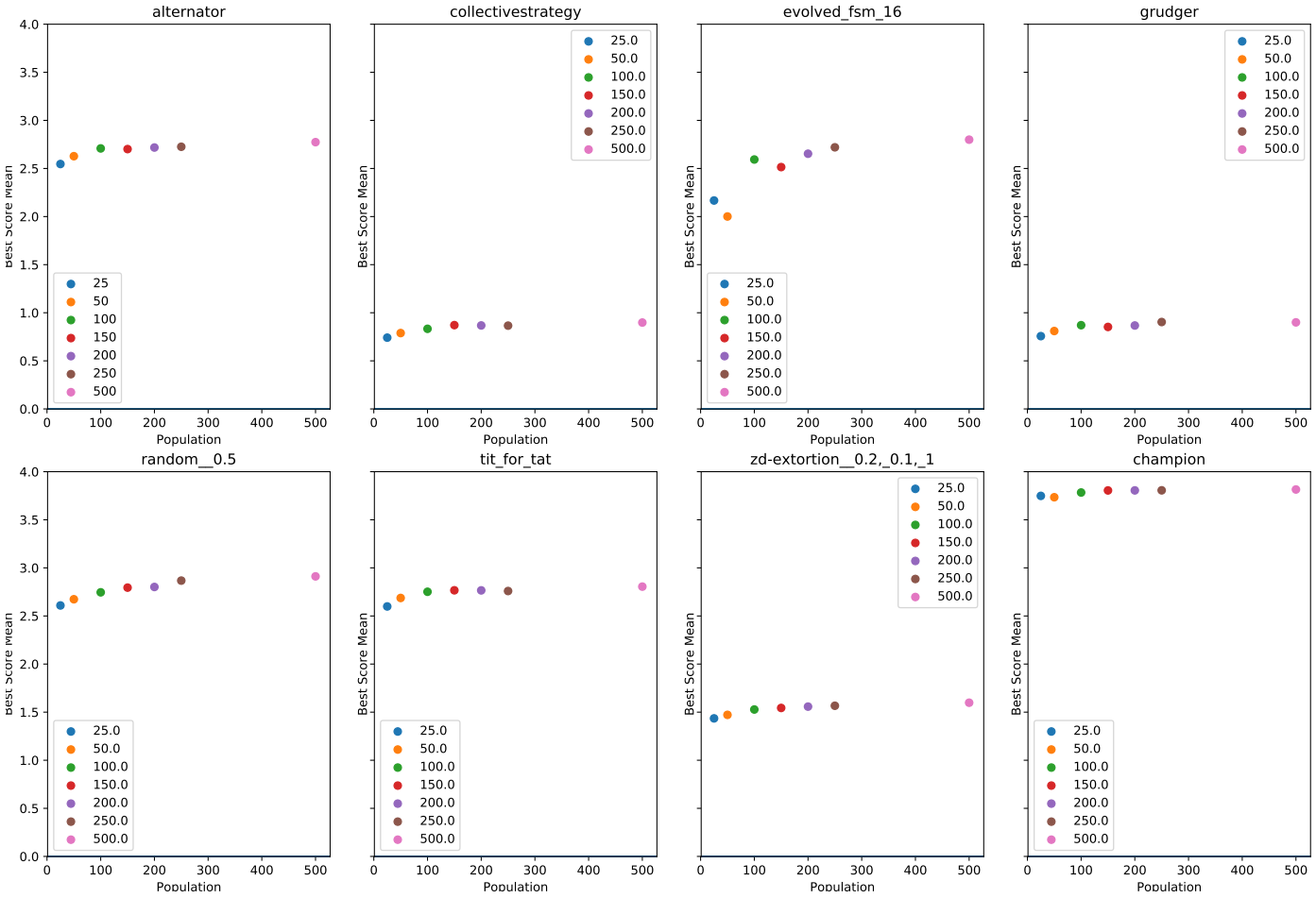

Figure 5.3: Scatter of max best score vs different initial populations

we would be more focused on what happens to certain averages of results across the whole run, rather than absolute improvement. If we look at figure 5.5, mean best score difference against the number of generations, we can observe how, on average, the number of generations has a declining effect the overall change in our best score per generation.

This mean increase of score per generation trend is to be expected; when we are close to a maximum it is more difficulty to randomly select which element in the sequence needs changing to improve our score. On this result we can conclude as we increase generations there is less and less benefit per generation. There is, however, still a benefit to extending the generations but we may have better performance by altering another parameter of the algorithm. There may be a benefit from increasing the mutation rates when we get close to one of these maximums; the more noisy our algorithm is for sequences could improve our chance of finding the correct solution. The probably of finding a solution as we narrow in on a maximum decreases due to the number of elements that, when changed, will provide a better score. Increasing the mutation frequency at this point means that there will be more members of the population that could potentially mutate the elements needed to improve the sequence.

Figure **??** shows the proximity the optimal solution sequence once the analysis has concluded. A good score is a score of 3 or more; this can change from player to player, and is never explicitly obvious. We can see that after a number of generations that solutions sometimes get 'stuck' in a local maximum score.

After 250 generations we *seem* to have reached a solution state for our opponents Tit for tat and alternator but not for grudger. Against Grudger we see an example, we have only reached an average score per turn of ___, which is obviously far from its optimal sequence. From the combination of the plots, having more generations means that there is, on average, less of an improvement per generation. It is clear that a higher number of generations is required to find a better solution sequence for an opponent. From now on, 250 is the number of generations we will use to find our solution sequence during the analysis.

```
def generationSizeChecker(opponent):
    file_name = "data/" + str(opponent) \
        .replace(" ", "_") \
        .replace(":", "_"). \
        lower() + "_generation.csv"
    if not os.path.isfile(file_name):
        df_main = pd.DataFrame(data=None, columns=col_names)
        for gens in generation_list:
            start_time = time.clock()
            pop_run = runGeneticAlgo(opponent,
                                     population_size=150,
                                     number_of_game_turns=200,
                                     cycle_length=200,
                                     generations=gens,
                                     mutation_probability=0.1,
                                     reset_file=True)
            end_time = time.clock()
            tmp_df = pd.read_csv(pop_run[0], names=col_names)
            tmp_df["generations"] = gens
            tmp_df["time_taken"] = end_time - start_time
            tmp_df["opponent"] = str(opponent)
            tmp_df["best_score_diff"] = np.append([0], np.diff(tmp_df["best_score"]))
            df_main = df_main.append(tmp_df, ignore_index=True)
        df_main.to_csv(file_name)
        print("List Complete:", file_name)
        return df_main
    else:
        print("file ", file_name, " already exists, no calcs to do.")
    file_df = pd.read_csv(file_name)
    # remove first column
    file_df = file_df[list(file_df)[1:]]
    return file_df
```

Figure 5.4: code to check multiple generation lengths.

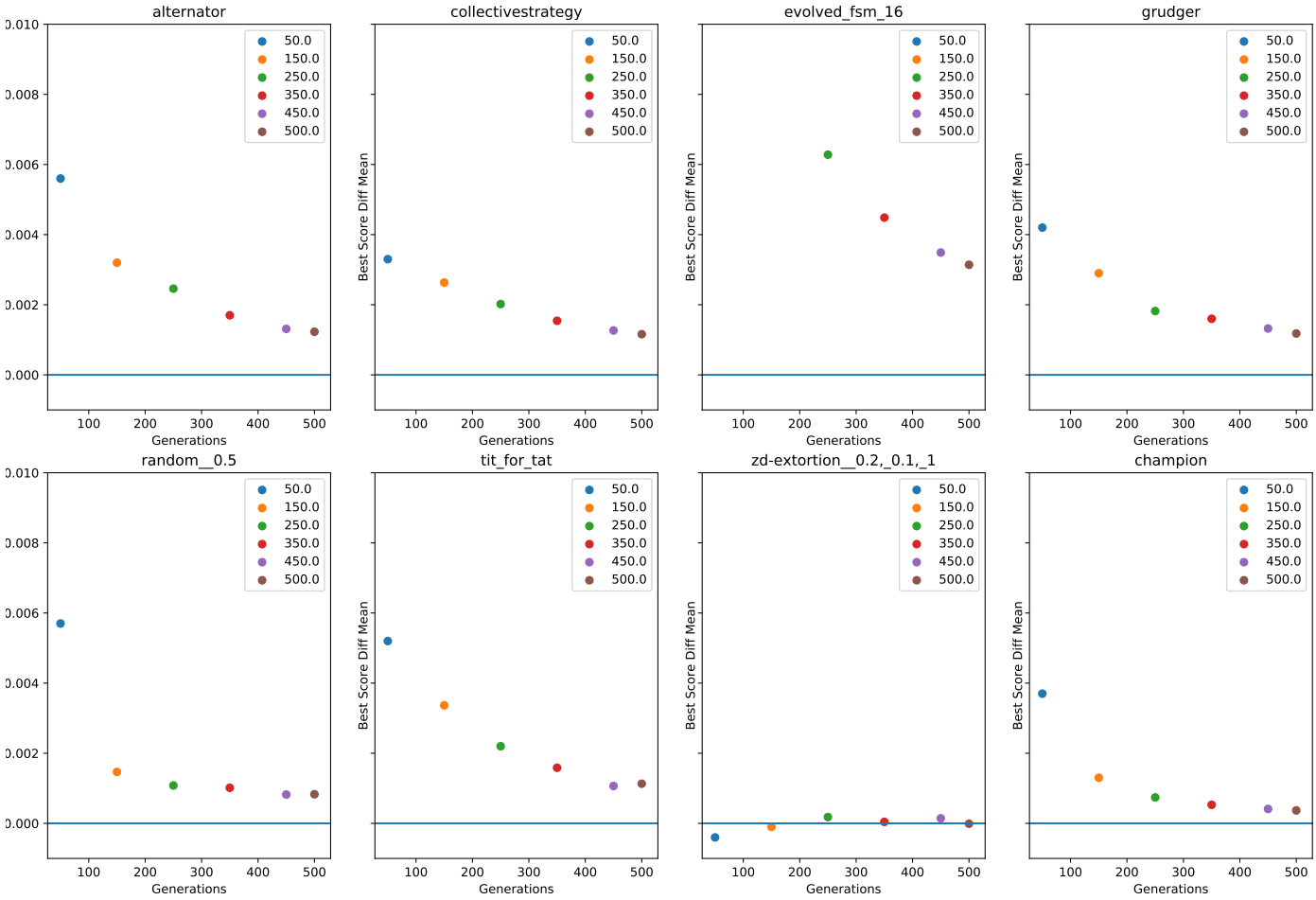

Figure 5.5: Mean Best Score diff vs total generation lengths

For most of the opponents 250 generations seems reasonable to reach a solution sequence as shown in the Alternator and Tit For Tat. However, there are clear signs of local maximums occurring in the Grudger example. Figure **??** has reached a better sequence in 450 generations than 500[3]; meaning that increasing the generation length doesn't necessarily mean a local maximum. The complexities with local maximums during the generations lie with mutation rates and crossovers. We will cover this in Section 5.4

In this investigation we will want to find the optimal solution and so, from these results, we will want to extend the generation length as far as possible. An infinite number of generations would be preferable, but we don't have an eternity so a selection of a relatively large generation size will be adequate when coming to the final series of tests.

5.3 Changing Mutation Rate

By changing the way in which we mutate our elements within a sequence, we might be able to more effectively narrow in on an optimal solution sequence. The default settings are a frequency of 0.1; meaning for every 10 members of our population that continue into the next generation one of these has some elements in its sequence changed, and a potency of 1; meaning that every sequence that was altered only has 1 element altered. Here we will look into these 2 different concepts and see how they might improve our distance to an optimal sequence and whether we can escape local maximums.

- Is it beneficial for more/less than 1 in 10 members to be mutated generation to generation? (More frequent mutation)

[3]These are independent trials and have different sequences.

```
def mutate(self):
    # if the mutation occurs
    if random.rand() <= self.mutation_probability:
        mutated_sequence = self.get_sequence()
        for _ in range(self.mutation_potency):
            index_to_change = random.randint(0, len(mutated_sequence))
            # Mutation - change a single gene
            if mutated_sequence[index_to_change] == C:
                mutated_sequence[index_to_change] = D
            else:
                mutated_sequence[index_to_change] = C
            self.sequence = mutated_sequence
```

Figure 5.6: The mutation code as given in the axelrod-dojo

- Is changing one or more actions of a members' sequence the best way of mutating a candidate (More potent mutation)

These are two separate questions, so first we will look at increasing the potency of our mutation. Once we have found some information on how this effects our solution, we can look into the frequency of our mutations with the new potency as a permanent setting. As shown further on, there is not much of an improvement on our algorithm to changing either of these.The mutation algorithm is shown in Snippet 5.6

EFFICIENCY NOTE: This approach allows for an $O(1)$ factor of scaling. This makes changes in mutation a great candidate for an approach to reduce our solution sequence distance compared with other approaches, for example increasing the population size.

5.3.1 Changing Mutation Potency

Changing the potency of the algorithm will mainly generate the noise in our sequence generation to generation, increasing the distance[4] between the mutated sequence from the original.

This potentially could create an algorithm that is too 'jumpy' for narrowing in on a solution. We can imagine a sequence as a vector in 200 dimension space then a mutation for element S_i is the same as changing the vector in its i^th dimension. Shortening this example to a vector in 3 dimensions (or a sequence of length 3) then a mutation is much more easily visualised. It is clear that a mutation potency should be kept low as to keep consecutively mutated sequences more similar; we will only be looking at mutating our sequences at up to 10 percent of their elements. We will look into having mutation potencies $M_p \in [1, 2, 3, 5, 10, 15, 20]$.

Using the data generated from Snippet 5.7 of code we are able to look at how our best score and our best score diff is affected as we increase the number of positions.

Figure 5.8 shows no clear benefit from increasing the mutation potency. We can see that having changed 15 genes in our sequence each time we are still not improving our score as much as changing only 1. This may be down to chance (and if the test is rerun this may disappear), however looking at more opponents than just Grudger we find there is no clear benefit to increasing the mutation potency with respect to the overall best score value against an opponent. If we instead look at what our average increase of score per mutation is we may observe a useful result.

From looking at how our average best score difference changes as we increase the mutation potency there is no sign that there is a significant improvement to our sequence. The increase in mean best score difference is not substantial and could be down to chance.

[4]Distance concept has been taken from coding theory; $d(s_1, s_2)$ = the number of differing positions between 2 sequences s_1 and s_2. For example: $d(111, 110) = d(CCC, CCD) = 1$

```
def mutationPotencyChecker(opponent):
    file_name = "data/" + str(opponent) \
        .replace(" ", "_") \
        .replace(":", "_") \
        .lower() + "_mutation_potency.csv"
    if not os.path.isfile(file_name):
        df_main = pd.DataFrame(data=None, columns=col_names)
        for potency in mutatuon_potency_list:
            start_time = time.clock()
            pot_run = runGeneticAlgo(opponent,
                                     population_size=150,
                                     number_of_game_turns=200,
                                     cycle_length=200,
                                     generations=250,
                                     mutation_probability=0.1,
                                     mutation_potency=potency,
                                     reset_file=True)
            end_time = time.clock()
            tmp_df = pd.read_csv(pot_run[0], names=col_names)
            tmp_df["mutation_potency"] = potency
            tmp_df["time_taken"] = end_time - start_time
            tmp_df["opponent"] = str(opponent)
            tmp_df["best_score_diff"] = np.append([0], np.diff(tmp_df["best_score"]))
            df_main = df_main.append(tmp_df, ignore_index=True)
        df_main.to_csv(file_name)
        print("List Complete:", file_name)
        return df_main
    else:
        print("file ", file_name, " already exists, no calcs to do.")
        file_df = pd.read_csv(file_name)
        # remove first column
        file_df = file_df[list(file_df)[1:]]
        return file_df
```

Figure 5.7: Mutation potency code

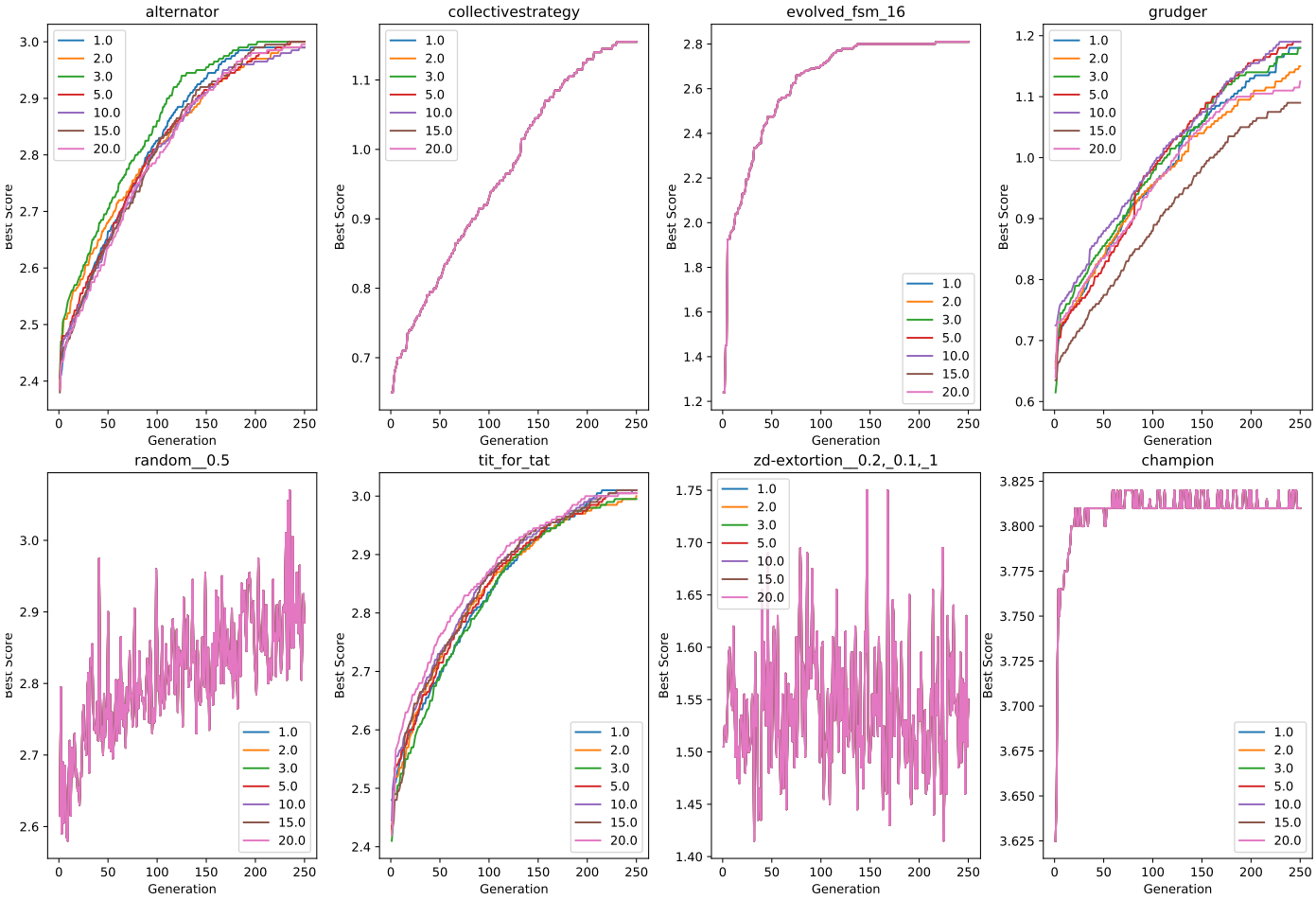

Figure 5.8: Best score vs generation for different mutation potencies

5.3.2 Changing Mutation Frequency

In contrast to changing the mutation potency, increasing the frequency should allow us to generate more unique sequences generation to generation. We will look at what happens when we run the genetic algorithm on a set of mutation frequencies $M_f \in [0.1, 0.2, 0.3, 0.4, 0.5]$. The code in Snippet 5.10 shows the algorithm that will complete this analysis.

The results on changing the mutation rate don't obviously effect that generations until convergence from this overview. There is an interesting result that can be seen on the grudger plot; the algorithm has found 2 different maximums.

From Figure 5.11 we can see that the mutation frequency of 0.1 and 0.2 produced higher scoring solutions than the other mutation frequencies. This shows that we have found 3 different solution sequences (in freqs .15, .2, .25) with the solution for .1 continuing to improve as the generations ended.

As an additional point, we can also observe that increasing the mutation frequency means that there is less variation in the best scoring sequences.

5.4 Mitigating local maximum solutions

The occurrence of local maximums is something that has only occurred in sequences against for the Grudger opponent so far. Figure 5.11 shows that there are clearly 2 distinct plateaus that are reached in terms of score, and the overall scores are much lower than the optimal, around 1.4.

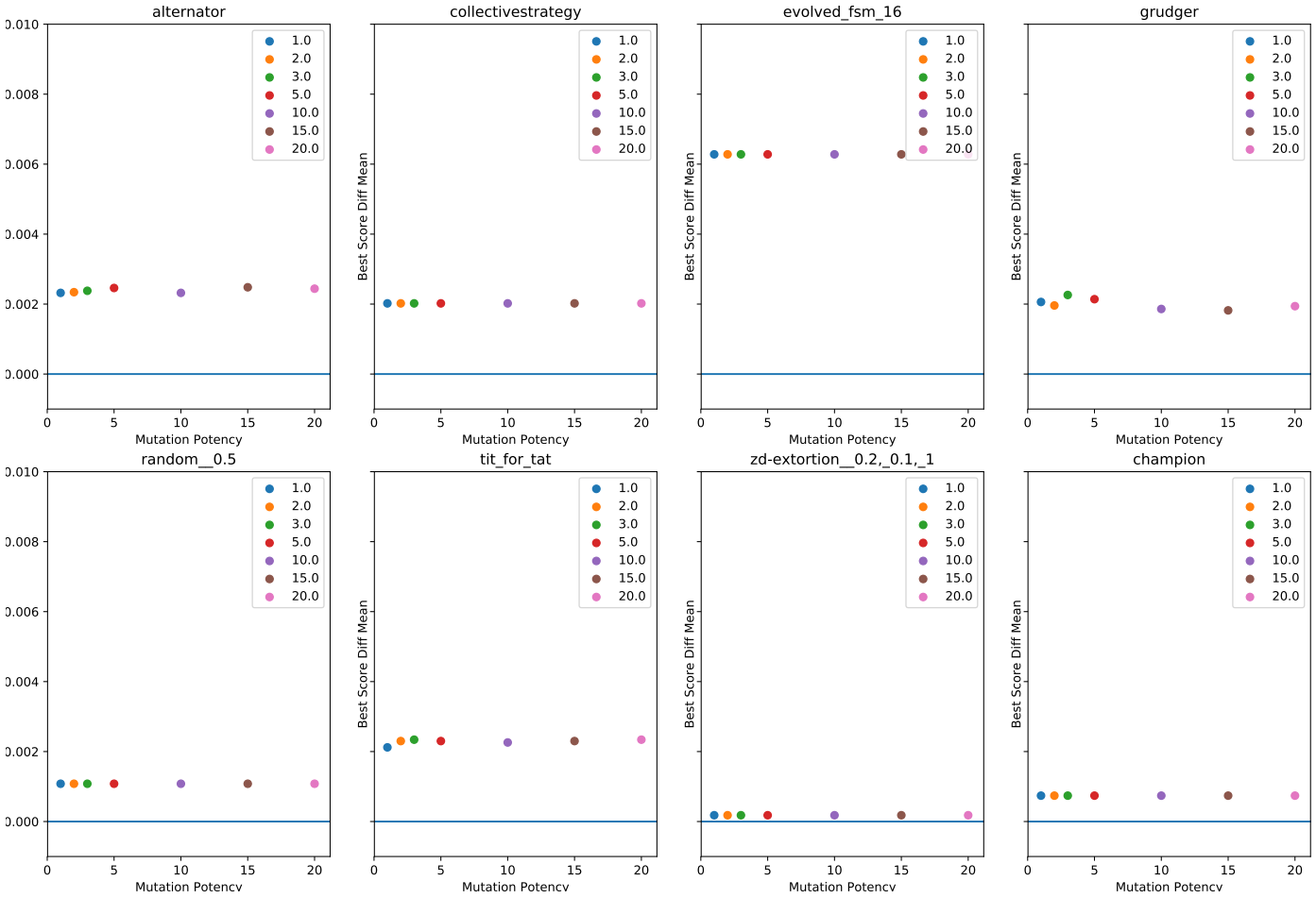

Figure 5.9: Average best score diff vs mutation potencies

The difference between the Grudger and the other opponents have been looking at is that the Grudger has a singularity point where its behaviour changes. The change in behaviour is not uncommon, Tit For Tat works in a similar way and in both cases our algorithm has managed to identify this behaviour and adapt to overcome its negative effects.

Grudger is an opponent which it is possible to attain a local maximum score and be 'trapped' in this solution sequence. From the output sequences below it appears that the sequence is being changed into becoming 2 sections of opposing moves. If we look at a random start sequence and then the end sequence after 250 generations we can see that the genetic algorithm is learning defect after some point and cooperate before. This is due to the fact a good solution will end in lots off defections after it has already defected, but cooperate beforehand to avoid this harsh behaviour.

Grudger best start: C6, 8, 2, 1, 2, 1, 6 . . . (No pattern is obvious.)

Grudger best end: $C : [22, 178]$

We may be able to identify a way to mitigate this singularity effect by looking at the difference between 2 similar solutions. Grudger and Tit For Tat differ in their strategies in two ways:

- Grudger never changes its mind. There is one change in behaviour for the entire game, unlike Tit For Tat. The algorithm then only has a single opportunity to observe this per population per generation meaning the behaviour is much less frequently encountered.
- Grudger will not become 'smart' again. This means that the move genetic algorithm picks up the effect of a single defection in its sequence it starts playing a 'dumb' opponent. A random start of Cs and Ds puts the likelihood of at least 1 defection occurring in the first 10 moves at above 99.99%. This swap from 'smart' to 'dumb' will, most

```
def mutationFrequencyChecker(opponent):
    file_name = "data/" + str(opponent).replace(" ", "_") \
        .replace(":", "_") \
        .lower() + "_mutation_frequency.csv"
    if not os.path.isfile(file_name):
        df_main = pd.DataFrame(data=None, columns=col_names)
        for freq in mutation_frequency_list:
            start_time = time.clock()
            pot_run = runGeneticAlgo(opponent,
                                     population_size=150,
                                     number_of_game_turns=200,
                                     cycle_length=200,
                                     generations=250,
                                     mutation_probability=freq,
                                     mutation_potency=1,
                                     reset_file=True)
            end_time = time.clock()
            tmp_df = pd.read_csv(pot_run[0], names=col_names)
            tmp_df["mutation_frequency"] = freq
            tmp_df["time_taken"] = end_time - start_time
            tmp_df["opponent"] = str(opponent)
            tmp_df["best_score_diff"] = np.append([0], np.diff(tmp_df["best_score"]))
            df_main = df_main.append(tmp_df, ignore_index=True)
        df_main.to_csv(file_name)
        print("List Complete:", file_name)
        return df_main
    else:
        print("file ", file_name, " already exists, no calcs to do.")
        file_df = pd.read_csv(file_name)
        # remove first column
        file_df = file_df[list(file_df)[1:]]
        return file_df
```

Figure 5.10: Mutation potency code

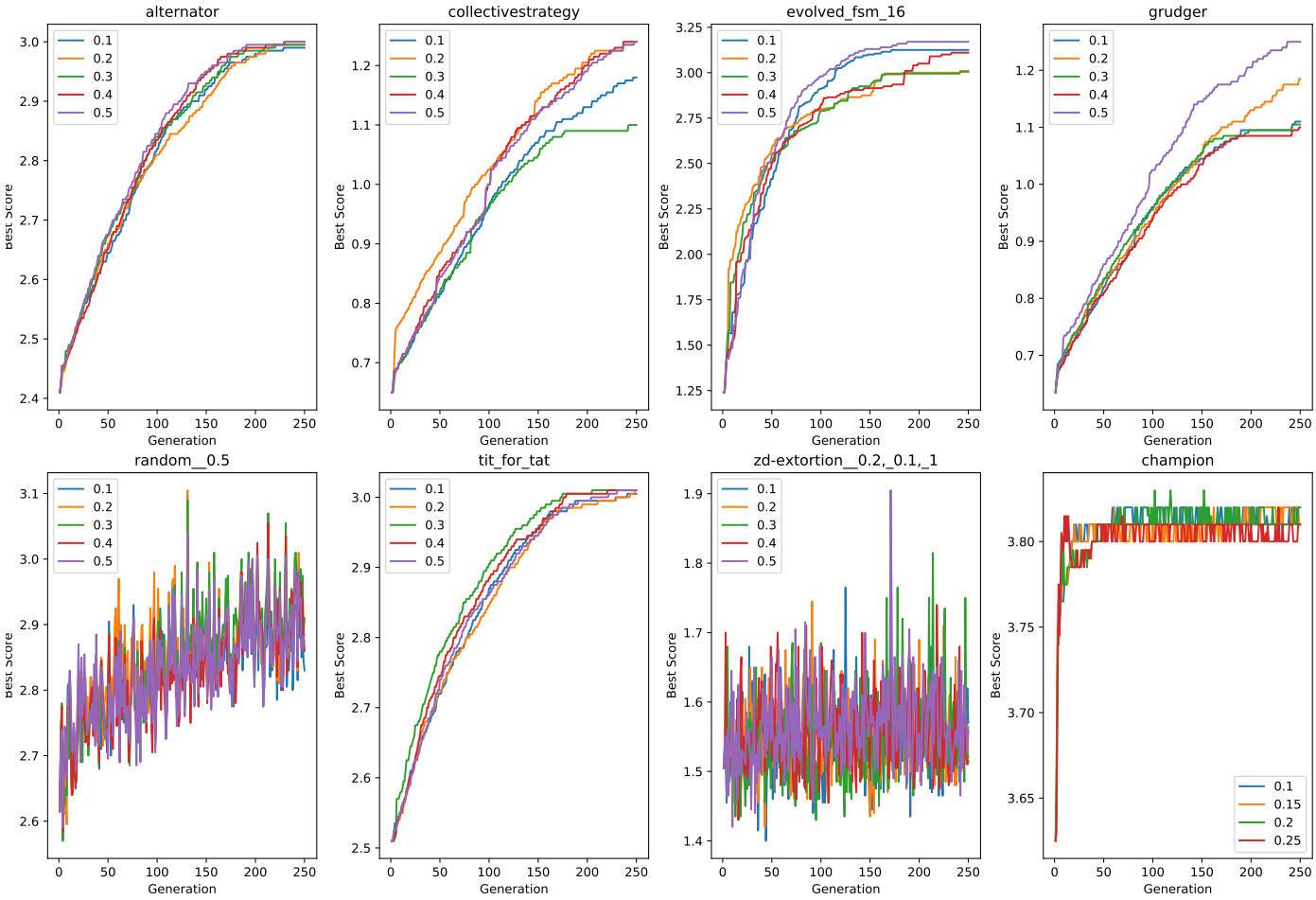

Figure 5.11: Best score vs generation for different mutation frequencies

likely, always occur in the first 10 moves; the only way of extending the 'smart' player is to add a cooperation to the end of the starting Cs.

Snippet 5.13 shows two totality games against grudger, one of all Cs and one of all Ds. These are edge cases and would be incredibly rarely encountered as a starting point in the random initial population. Because of this, the algorithm has to shuffle towards the potential benefit of using these totalities rather than start with analysing them. Our case against grudger requires the algorithm to attempt this shuffle towards a totality after encountering the grudging effect. This would then require the algorithm to select (using random selection out of 200) the first defection move and change it (only 1 in 10 members are mutated) to a cooperation move. This likelihood is incredibly small, and starting at common or uniform sequences would be more beneficial, as posed in Section 5.5.

As Snippet 5.13 shows, we should be converging on a totality of Cs rather than what its doing; finding the totality of Ds. This is probably because the algorithm initially limits its best score per turn once the first generation is complete and a cut-off has been established for each of the initial population. The crossover method between generations then doesn't provide enough of a mix up to allow the algorithm to escape the local minimum by switching a subsection with a sufficiently different potentially better subsection. Then when it comes to mutating, there is little any number of mutations can do to drastically change large sections of the sequence without having a huge effect on the score.

5.4.1 Ineffective Approach of Altering Crossover And Mutation

The process of converging to Ds when building a solution against grudger then sheds light on the process the algorithm takes to find a solution. If we are to find the optimal solution sequence, we must take a crossover and mutation path which

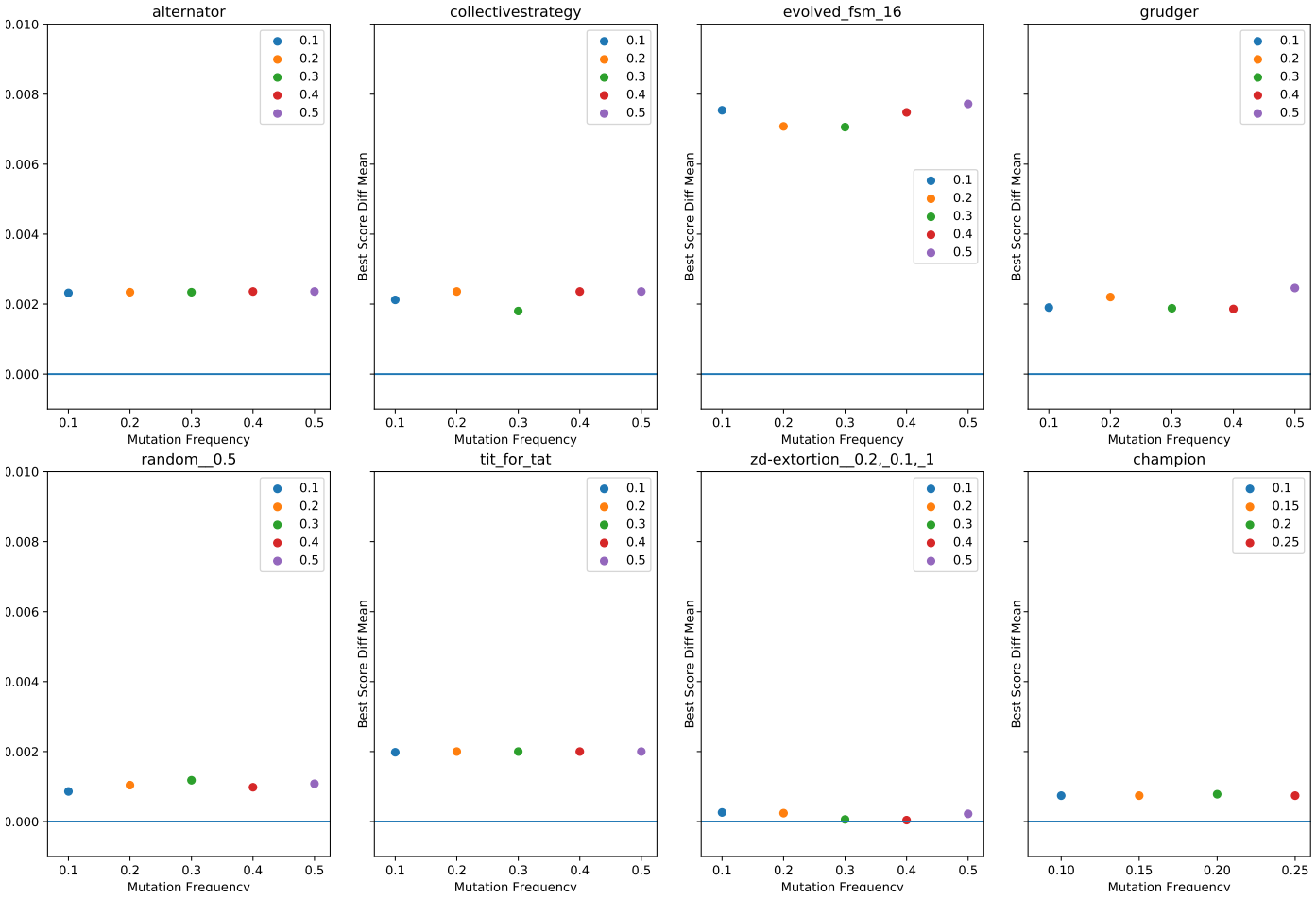

Figure 5.12: Average best score diff vs mutation frequencies

doesn't cut off better paths as we work our way towards good solutions; this is much easier said than put into practice due to the way the algorithm 'cuts off paths'. If we reverse this thinking and try to alter our crossover design and mutation rate such that instead of 'cutting off' a path we are able to 'build' new ones. We can re-design the crossover to switch up large subsections of the sequence then allow the mutations to optimise these sub-sequences.

Currently we have the following design:

We want to allow the crossover to have more of an impact than just halving the sequence and optimizing each section.i.e.go from:

$$|----------\text{-}|and|++++++++++++++++++++|$$
$$=|----------++++++++++|$$

to the mixture:

$$|----------\text{-}|and|++++++++++++++++++++|$$
$$=|--++--++--++--++--++|$$

This will allow the mutation rate to edit the subsections in a more interlaced manner, hopefully overcoming the pitfalls of sparse mutations to escape local maximums. Our new crossover method is shown in Figure 5.15. As shown, the algorithm splits the two sequences into 10 section and the new sequence is formed from Alternating sections. Figure 5.16 shows an example of the new crossover method.

We will look at how this new crossover algorithm works with the default mutation (freq=.1 and pot=1) to improve our local maximums with the selection off opponents.

```
players = (axl.Grudger(), axl.Cycler("C"))
match = axl.Match(players, 200)
match.play()
print("final scores:", match.final_score())
print("per turn:", match.final_score_per_turn())

# >final scores: (600, 600)
# >per turn: (3.0, 3.0)

players = (axl.Grudger(), axl.Cycler("D"))
match = axl.Match(players, 200)
match.play()
print("final scores:", match.final_score())
print("per turn:", match.final_score_per_turn())

# >final scores: (199, 204)
# >per turn: (0.995, 1.02)
```

Figure 5.13: Grudger matches against totalities

```
def crossover_old(self, other_cycler,in_seed=0):
    seq1 = self.sequence
    seq2 = other_cycler.sequence

    if not in_seed == 0:
        # only seed for when we explicitly give it a seed
        random.seed(in_seed)

    midpoint = int(random.randint(0, len(seq1)) / 2)
    new_seq = seq1[:midpoint] + seq2[midpoint:]
    return CyclerParams(sequence=new_seq)
```

Figure 5.14: Old Crossover algorithm

```
def crossover(self, other_cycler):
    # 10 crossover points:
    step_size = int(len(self.get_sequence()) / 10)
    # empty starting seq
    new_seq = []
    seq1 = self.get_sequence()
    seq2 = other_cycler.get_sequence()
    i = 0
    j = i + step_size
    while j <= len(seq1) - step_size:
        new_seq = new_seq + seq1[i:j]
        new_seq = new_seq + seq2[i + step_size:j + step_size]
        i += 2 * +step_size
        j += 2 * +step_size
    return CyclerParams(sequence=new_seq)
```

Figure 5.15: New Crossover algorithm

```
seq1 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
seq2 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
step_size = int(len(seq1) / 10)
i = 0
j = i + step_size
new_seq = []
while j <= len(seq1) - step_size:
    new_seq = new_seq + seq1[i:j]
    new_seq = new_seq + seq2[i + step_size:j + step_size]
    i += 2 * +step_size
    j += 2 * +step_size
print(seq1)
print(seq2)
print(new_seq)

# >[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
# >[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
# >[1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0]
```

Figure 5.16: Example of new crossover algorithm

There was no noticeable improvement from introducing this change of function. Each players average score per turn was in the expected score if the algorithms crossover method was unchanged. The problem of mitigating sub optimal solutions may be more efficiently solved using a predefined population. Section 5.5 discusses this in more detail.

5.5 Altering Initial Population

When performing analysis of an opponent using a GA it can sometimes be useful to brute force the starting positions of our feature selection to create rare[5] starting points. Altering the initial population allows us to select starting sequences that fit patterns we know will have good results. For example totalities, with heads/tails, are usually very effective against simple opponents; for example Tit For Tat, 'dumb' opponents or Grudger. Alternating and certain known solution optimal sequences are good starting sequences for an initial population, allowing a more intelligently distributed set of starting points for the random mutation process.[6]

This section discusses the results of working on an initial population that contain common solution sequences. We will start by creating a population of 'neat' starting members then allow the entropy of the genetic algorithm to alter these sequences. Deciding where to start our algorithm may mitigate potential sub-optimal solutions by reducing the distance between the starting sequences and optimal solutions. The list of 'neat' starting points are stated below:

Totalities

- $C : [200]$ — 1 sequence

Single Change Sequences

- $\{C : [i, 200 - i]\} \quad i \in [1, 10],\ i \in \mathbb{Z}$ — 10 sequences
- $\{C : [200 - i, i]\} \quad i \in [1, 10],\ i \in \mathbb{Z}$ — 10 sequences

Matching Tail Sequences

[5] For example of a random sequence that is a totality of Cs is incredibly rare, with around 6.2^{-61} chance of occurring naturally.

[6] This can be visualised as placing balls on a 'lumpy' 3d plane to try and find the minimum, starting with an educated guess means we wont get all the balls stuck in one valley which isn't the deepest.

- $\{C : [i, 200 - (i + j), j]\} \quad i, j \in [1, 5],\ i, j \in \mathbb{Z}$ — 25 sequences

Alternating

- $C : [\ (i, i)^{100/i}] \quad i \in \{1, 2, 4, 5\}, i \in \mathbb{Z}$ — 4 sequences

Handshakes

- $C : [\ i, j, k, 200 - (i + j + k)] \quad i, j, k \in \{0, 1, 2, 3\}, i \in \mathbb{Z}$ — 4 sequences

For each of these sequence sets the inverse, with a defection move start, will also be added to the set of starting sequences. This in total gives 164 (after code 5.18) of sequences, we will then make up the difference to the population limit using a set of random sequences.

We can now look into how this has effected the previous experiments.

5.5.1 Population Size

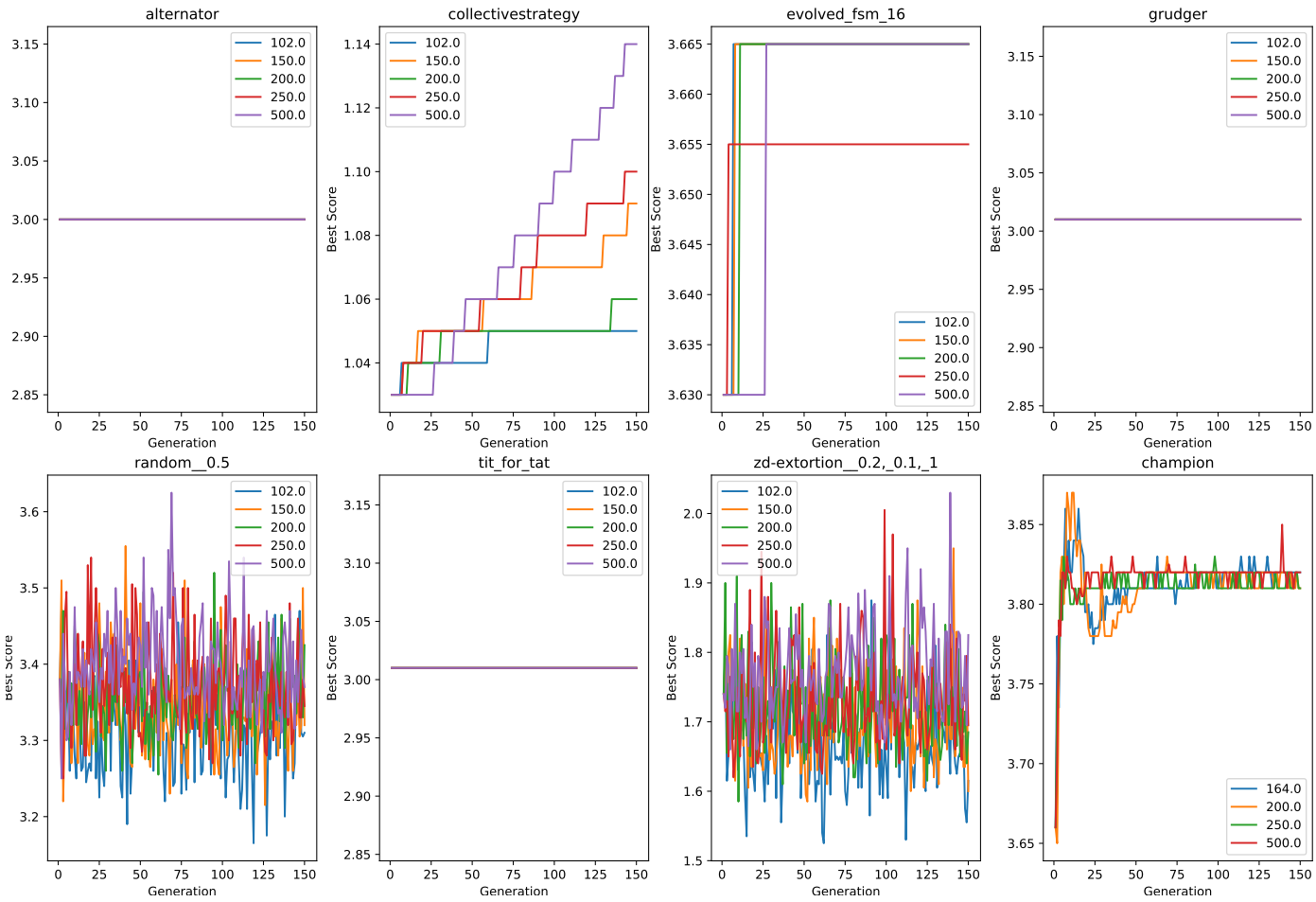

Figure 5.17: Best Score vs generations for pre-set initial populations on top of random sequences

Figure 5.17 shows the results of using a given initial population of sequences of size 102 along with supplementing this with random members. We will have $|P| \in [102, 200, 250, 500]$. The Collective Strategy run in figure looks as if there are

much better results with an initial set population for the algorithm. Of the two that are not showing straight lines we can observe that the algorithm has found the optimal sequence for all but Random & Collective Strategy. This is probably due to 2 different reasons:

Random: The 'dumb' strategy should being beaten with a totality of D, the algorithm has in fact converged to almost this totality, but still has intermittent Cs. The reason for this is the scoring grade 'score per turn' will reflect on the number of intermittent Cs in the Random players sequence. More Cs in the Random sequence will allow the algorithm to score more. This leads to a solution sequence containing some random Cs not because they score better in some turns, but because the totality of Ds played against a Random sequence would 'loose' to a sequence containing some Cs in because its not a fair trial; the two sequences play against different Random opponent sequences. **Collective Strategy**: as described in chapter 2.3 this strategy is, basically, a handshake + Grudger. If we look into the sequence it selected we can see its found the handshake but then arrives on Grudger. After this encounter the same problem as we had before population selection occurs and the algorithms limits the damage by splitting into Cs then Ds.Solving the collective strategy (and handshakes in general) may be simple; we just put in all the possible n move handshakes followed by totalities and then set to work on the 2nd part of the sequence.

It is clear that having a larger population is good from the old analysis, but the initial population still has to be tweaked to improve its scores against certain handshake opponents. The Random opponent is a special case that requires more analysis to find the absolute optimal. The new initial population, now of size 164, will include all combinations of C & D of length 5, followed by finishing on all Cs or Ds as shown in the Totalities & handshakes section of Snippet 5.18. This run will also supplement with random members for running test for $|P| \in [164, 200, 250, 500]$

After adding the extra members of the initial population we can see, from Figure 5.19, that now the handshake strategy (such as Collective Strategy) are solved much sooner. Also shown in Figure 5.19 is the Random and ZD extort players, these are examples of Stochastic players which are not finding optimal strategy's. These will be discussed further in Sections 5.5.4& **??**.

5.5.2 Generation Length

Figure 5.20 shows the results from changing the number of generations the algorithm will run to $G \in [50, 150, 250, 350, 450, 500]$ whilst also running with a population size of $|P| = 200$.

The Figure 5.20 shows no real improvement from increasing the generations, we will use the results from analysis above sections for constructing the full analysis.

5.5.3 Mutation Potency

Figure 5.21 shows the results from changing mutation potency the algorithm will use to be $M_p \in [1, 2, 3, 5, 10, 15, 20]$ whilst also running with a population size of $|P| = 200$.

The Figure 5.21 shows no real improvement from increasing this variable of the algorithm, we will use the results from analysis in previous sections for constructing the full analysis.

5.5.4 Mutation Frequency

Figure 5.22 shows the results from changing mutation potency the algorithm will use to be $M_f \in [0.1, 0.15, 0.2, 0.25]$ whilst also running with a population size of $|P| = 200$.

The Figure 5.22 shows no real improvement from increasing this variable of the algorithm, we will use the results from analysis in previous sections for constructing the full analysis.

```python
def getCyclerParamsPrePop2(pop_size=200, mutation_prop=0.1, muation_pot=1):
    pop = []
    if pop_size < 164:
        print("population must be 164+,")
        return

    # Totalities & Handshakes
    handshake_leng = 5
    for start in itertools.product("CD", repeat=handshake_leng):
        pop.append(axl_dojo.CyclerParams(list(start) + [C] * (200 - handshake_leng)))
        pop.append(axl_dojo.CyclerParams(list(start) + [D] * (200 - handshake_leng)))

    # 50-50
    pop.append(axl_dojo.CyclerParams([C] * 100 + [D] * 100))
    pop.append(axl_dojo.CyclerParams([D] * 100 + [C] * 100))

    # Single Change
    for i in range(1, 11):
        pop.append(axl_dojo.CyclerParams([C] * i + [D] * (200 - i)))
        pop.append(axl_dojo.CyclerParams([D] * i + [C] * (200 - i)))

    for i in range(1, 11):
        pop.append(axl_dojo.CyclerParams([C] * (200 - i) + [D] * i))
        pop.append(axl_dojo.CyclerParams([D] * (200 - i) + [C] * i))

    # Matching Tails
    for i in range(1, 6):
        for j in range(1, 6):
            pop.append(axl_dojo.CyclerParams([C] * i + [D] * (200 - (i + j)) + [C] * j))
            pop.append(axl_dojo.CyclerParams([D] * i + [C] * (200 - (i + j)) + [D] * j))

    # Alternating
    pop.append(axl_dojo.CyclerParams([C, D] * 100))
    pop.append(axl_dojo.CyclerParams([D, C] * 100))
    pop.append(axl_dojo.CyclerParams([C, C, D, D] * 50))
    pop.append(axl_dojo.CyclerParams([D, D, C, C] * 50))
    pop.append(axl_dojo.CyclerParams([C, C, C, C, D, D, D, D] * 25))
    pop.append(axl_dojo.CyclerParams([D, D, D, D, C, C, C, C] * 25))
    pop.append(axl_dojo.CyclerParams([C, C, C, C, C, D, D, D, D, D] * 20))
    pop.append(axl_dojo.CyclerParams([D, D, D, D, D, C, C, C, C, C] * 20))

    seq_len = 200
    while len(pop) < pop_size:
        random_moves = list(map(axl.Action, np.random.randint(0, 1 + 1, (seq_len, 1))))
        pop.append(axl_dojo.CyclerParams(random_moves))

    return pop
```

Figure 5.18: Initial Population Code

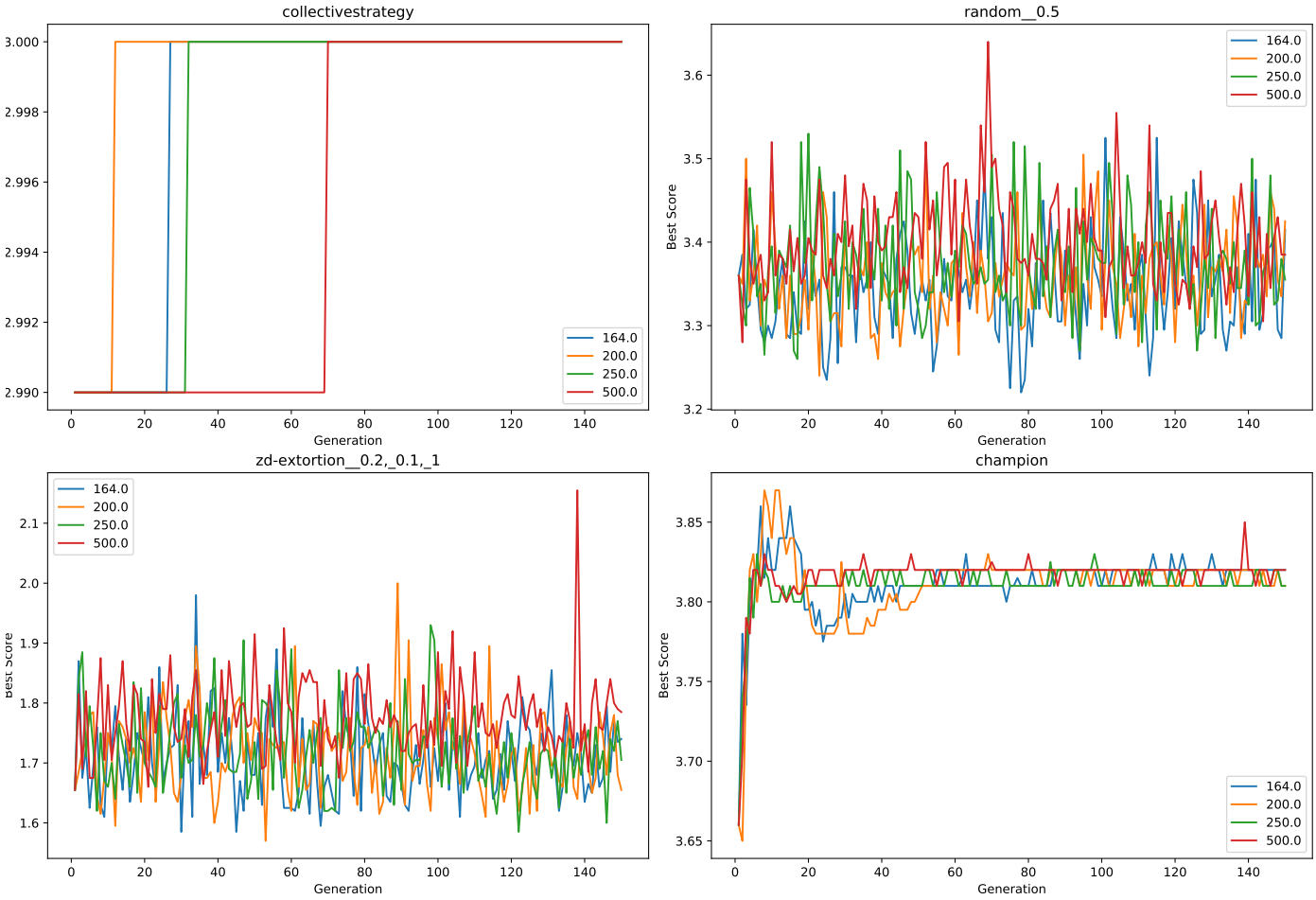

Figure 5.19: Non optimal sequence players after changing initial population

Discussion

The approach of adding a predefined set of members to the population before running the algorithm was incredibly successful. We have seen that for most opponents the solution sequence were looking for is given almost immediately. This result will mean that when calculating the solution sequence for the main population the predefined set, generated by Snippet 5.18, will be used to shorten analysis times.

Opponents such as Random, ZDExtort and Champion are showing that they're not finding the optimal solution. This is due to the fact these player belong to the class of stochastic opponents. Section 5.6 will look into these in more detail, putting forward possible approaches to simplify 'solving' these opponents.

5.6 Stochastic Opponents

Stochastic opponents form a problem with respect to continuity. The algorithm we're using continues to generate a new opponent sequence for every generation we run. This, essentially, leads to the algorithm playing a new version of the opponent every time, leaving very little opportunity to identify features which can be exploited.

The concept of a stochastic opponent can be somewhat 'overridden' by seeding the pseudo random number generator that creates the parameters that define what moves our opponent will take. Currently the only way of doing this is by seeing the whole Axelrod library upon initialising the opponent instance, Snippet 5.23 shows the code for wrapping any given opponent with the global seed command.

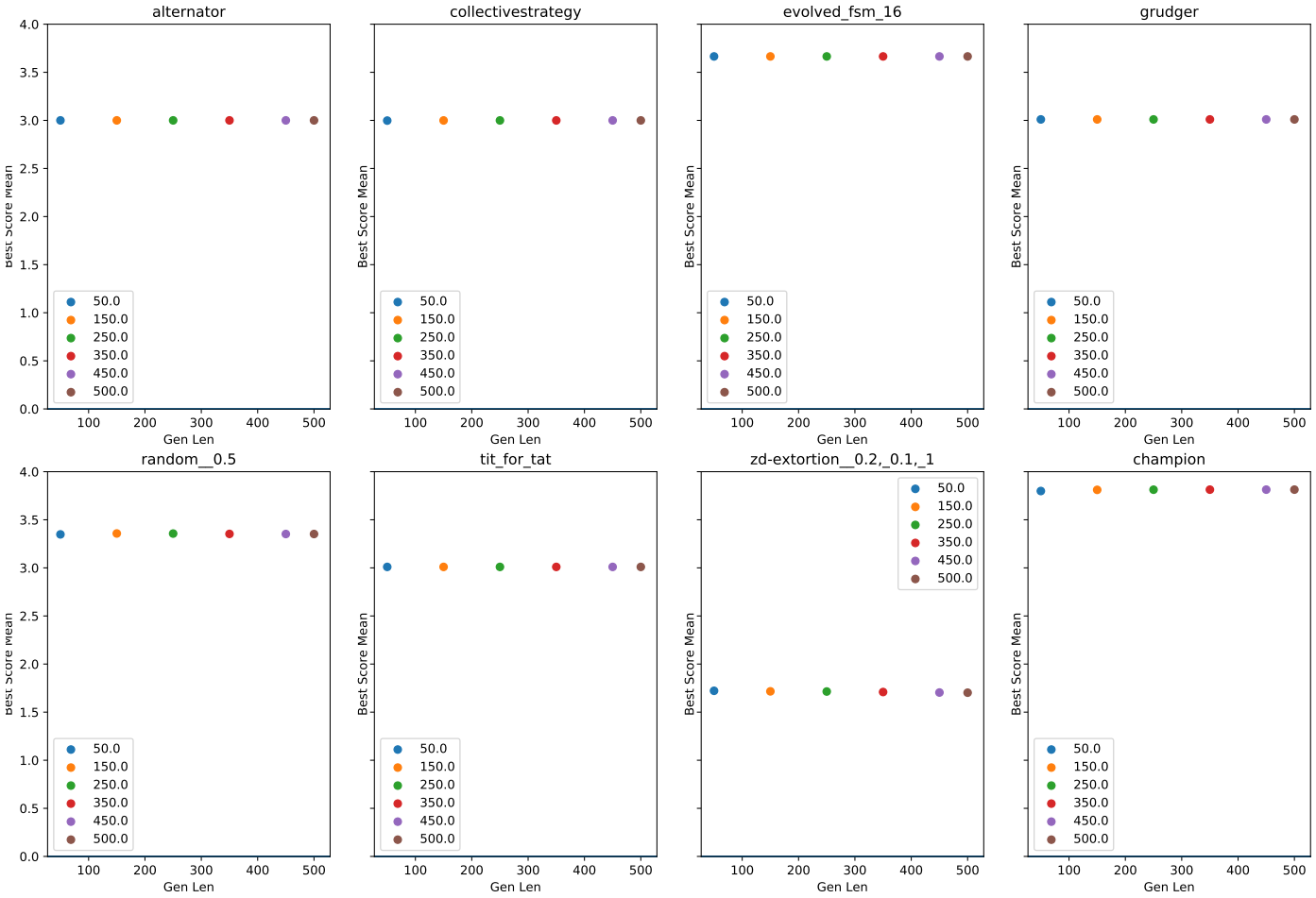

Figure 5.20:

When calculating the sequence solution for all of the opponents we will do so with seeded versions of the stochastic opponents. The parameters for stochastic opponents will be the default set by the creators and new instances will be made for any variations with real world applications. For example, default Random players will have a 0.5 chance of playing a C move; this parameter can optionally be changed, and will for certain opponents.

5.7 Conclusion of approach

After constructing the final analysis factory and running some environment checks we made the following changes to the code for ease of use:

- Number of generations changed: 300 → 600.
- Crossover algorithm, shown in Figure 5.15, was reverted to code given in Figure 5.14

These changes resulted in a streamlined piece of code we were able to run on the Cardiff University Mathematics department big compute machine. Each opponent was submitted to the compute engine using the standard analysis factory class `AnalysisRun`, shown in Figure B.2, using native multi-threading on a Linux OS to improve individual opponent analysis run times and the overall scalability of the project. We ran the code over a period of days resulting in 200+ output files of data which could then be analysed.

Reflecting on the execution and deployment of the code shows that we could have improved the analysis of short run

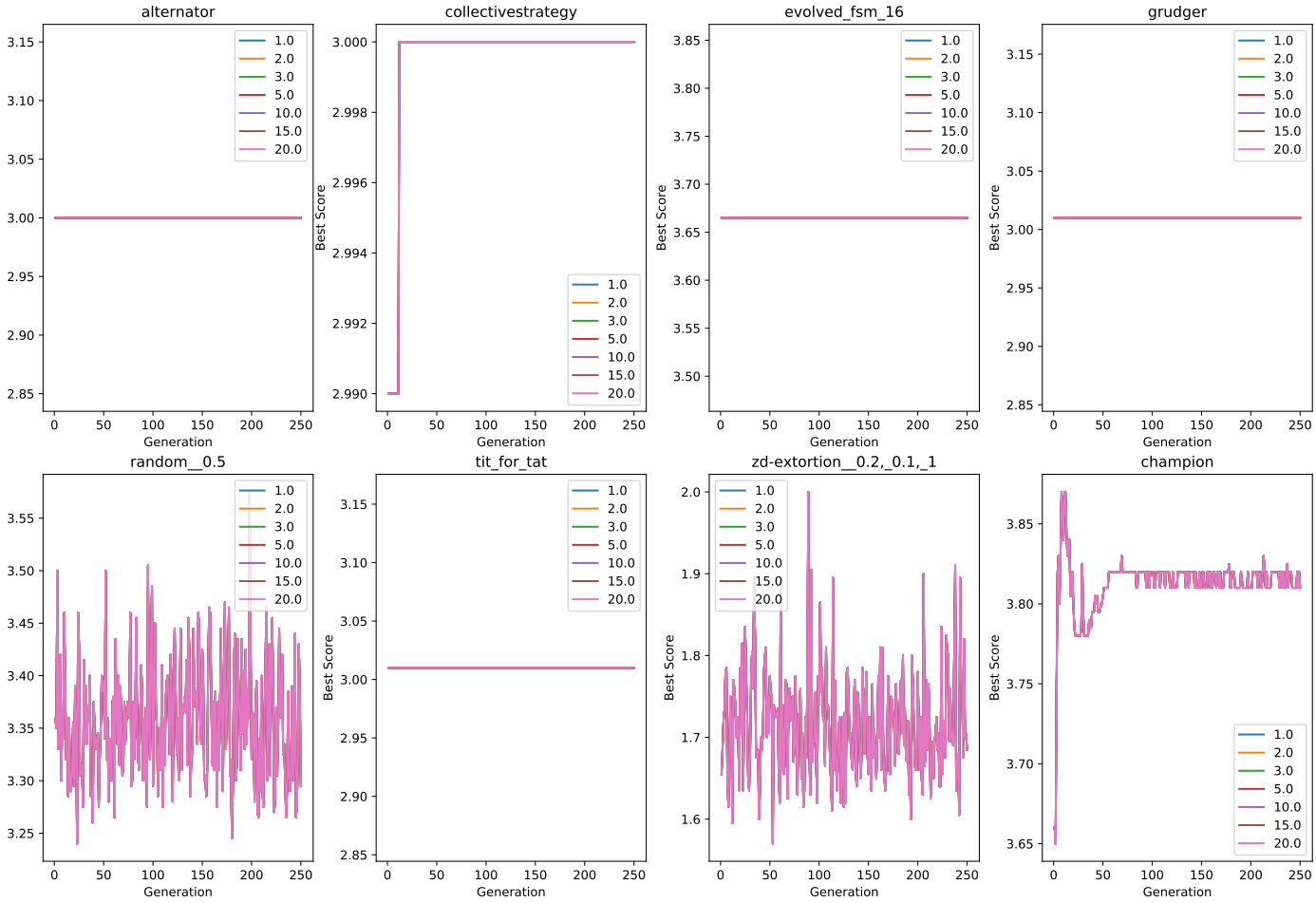

Figure 5.21:

time strategies against longer run time strategies and the reproducibility of the batch processing within the Axelrod Dojo Module. As far as each individuals analysis goes, I feel like the approach of adding predetermined sequences to an initial population was a huge improvement in compute time and overall predictions of best sequences.

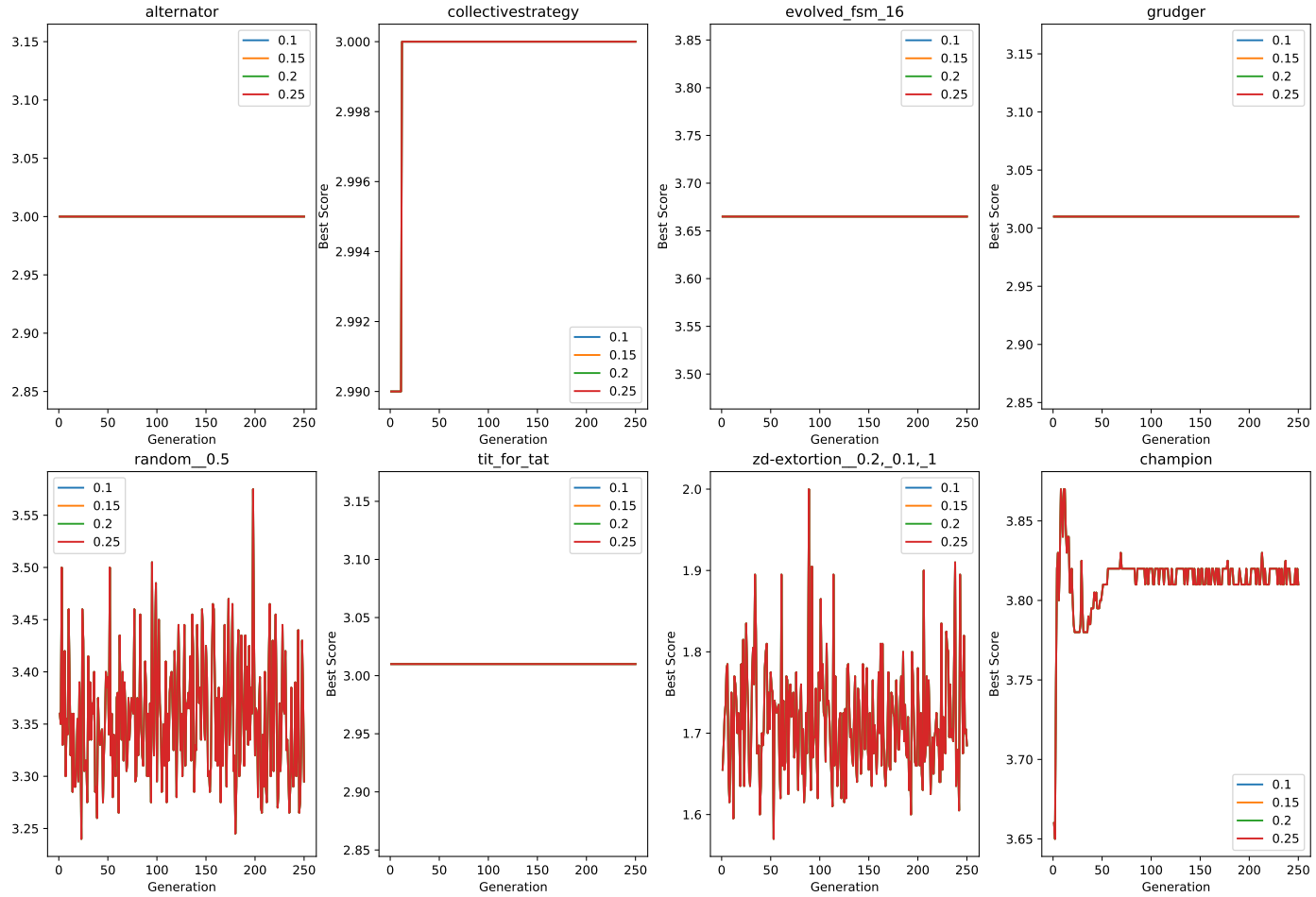

Figure 5.22:

```python
def getSeededPlayer(player_class):
    class NewClass(player_class):
        def __init__(self, seed=0):
            axl.seed(seed)
            super().__init__()

    return NewClass
```

Figure 5.23: A function for wrapping a player with a global seed function call

Chapter 6

Results and Discussion

This chapter will look into the results we gained after running the analysis across all opponents. As of writing this section there are strategies missing from the analysis due to run times. Details of the distribution of the results and basic output from the computation are given in section 6.1.

The analysis performed in this chapter contains data for opponents listed in Appendix C. Opponents without data due to being incomplete are all the (LRT) strategies listed. As these become available they will be added to the analysis.

6.1 Descriptive Data

The data we received from the final analysis contained 760 of the XXX opponents we set out to analyse. The missing data is due to long run time opponent with high computational complexity.

Figure 6.1 shows a raw data head from the output of the ϕ opponent.

gen	score mean	score median	score var	score range	best score	best sequence	name	seed
1	2.90966	3.0575	1.665496	4.985	5.0	DDD...	ϕ	0
2	4.79004	4.9250	0.420476	2.275	5.0	DDD...	ϕ	0
3	4.79352	4.97	0.489173	2.185	5	DDD...	ϕ	0
4	4.81386	5	0.460776	2.2	5	DDD...	ϕ	0
5	4.81826	5	0.479307	2.18	5	DDD...	ϕ	0
6	4.83508	5	0.430063	2.03	5	DDD...	ϕ	0
...	...	...	...	...	...	...	...	...

Table 6.1: Raw data from `AnalysisRun.py` output file

The data collected was, for each of the strategies, merged and collated with metadata to form an analysis table which was then processed. This metadata data included player classifiers as given in the Axelrod library. ultimately this data came down to one usable piece of information which was the stochastic boolean.

When analysing the overall best sequences for each opponent we can pick out some interesting and descriptive data. The distribution on best scores are showing in Figure 6.1, its kernel density estimate (KDE) exaggerates that fact we have a skew towards the higher scores with a fat tail on 4.5 to 5. This means, in a head to head, we can score higher than the majority of opponents to win the game. If there is a way of identifying an opponent then we would be able to tip the scales in a tournament in our favour by playing the sequences shown.

Figure 6.2 tries to show patterns related to how the nature of sequences change depending on score and opponent type. What can be seen is that there are lots of variation and not much in the way of a significant correlation between any of

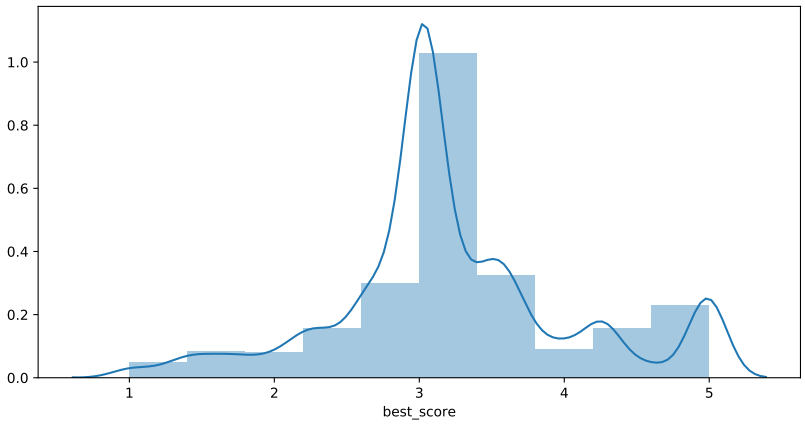

Figure 6.1: A histogram showing the distribution of best scores with overlaid KDE

the variables. Looking at the blank areas of the diagram shows more about the dat we have produced however. There are sections, namely low scoring & mid number of blocks for stochastic opponents dont overlap with non stochastic opponents. This can be explained by the fact there are lots more negative stochastic opponents, like ZD strategies and Q learners which would appear here.

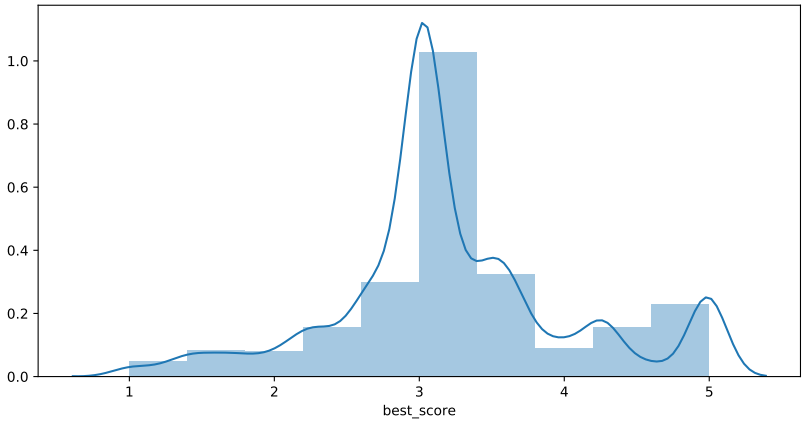

Figure 6.2: A joint plot of best score vs number of blocks coloured by stochastic boolean

6.2 Solution Distance Matrices

The first avenue of analysis, after constructing descriptive data, was to look at the relationship solutions have with each other. A distance matrix shows how much sequence for one opponent differs from every other, if 2 sequences are similar with respect to the distance function then they will score lower than 2 sequences that are more distinct.

In each matrix we order the sequences by score; $S_{O_0} \geq S_{O_1} \geq \ldots \geq S_{O_n}$, but we will shorten this notation S_{O_i} to S_i for simplicity. The solution sequence for the ith opponent, S_i vs the solution sequence for the jth, S_j is scored using our distance function, for example $d(S_0, S_n)$ is the distance between the best and worst scoring sequences respectively. The Matrix itself will be symmetric down the diagonal, so looking across rows vs columns tends to make more logical sense; the top rows are the highest scoring opponents, and the lower rows the worse scorers.

Hamming Distance

$$d(S_i, S_j) = S_i \cdot S_j^T \text{ or } 1 - \frac{\sum_{i,j=0}^{n} \delta_{ij}}{n} \text{ where } \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

The Hamming Distance represents the count of the elements that differ in any two sequences. A Hamming Distance will thus correspond to the number of places we have to play different moves against opponents to get out best score. Figure 6.3 shows the matrix generated by the code in Figure B.3.

By observing the graph row by row, we can build an idea of how similar each sequence is to the range of others. The top section of the plot shows the best scoring sequences vs other high scoring sequences on the left, and vs the worse scoring sequences on the right. At the very top left there crossing dark and light columns, suggesting that there are lots of very similar or dissimilar solutions at the high vs high end of the score level. As we move to the high vs low scores there are larger blocks of dark red representing high dissimilarity. This is shown again as we move to the bottom of the diagram. The large blocks or red covering the bottom 100 or so rows show that the lowest scoring opponents have very different sequences to the higher scorers, but as a group of low vs low they're quite similar.

We can look into what distances we expect by trying to analyse how scored are formed. The recording of an average score means that we can estimate the ratio of move combinations that formed a score because we know, if we are the first player, the pay offs for the combinations: $(C, C) = 3$, $(C, D) = 0$, $(D, C) = 5$, $(D, D) = 1$. Thus we can assume scoring in the midrange (3 ish) means mostly cooperating, or a combination of (C, D) and (D, C), whereas at the high and low ends its more likely we are to be defecting. This would mean when looking at the high vs low section and low vs high section of the graph we would expect a high level of similarity; $d(S_i, S_j) \approx 0$. However this hypothesis isn't supported, its clear there's a mix of distances because all 4 corners of the graph are not showing $d(S_i, S_j) \approx 0$.

To understand why this disparity between expectation and result exists we can look at a visual representation of each of the strategies. Figure 6.7 shows what the solution sequence looks like move by move, sorted by best scoring at the top of the left image moving down to worst scoring on the bottom of the right. From this diagram it is clear that some of the predictions above were true; the very high and low scoring solutions are totalities of defections. There is however much more noise for sequences at the top of the scoreboard, resulting in the large distances between the scores.

Figure 6.7 also shows some interesting results about picking up points in patterns. Some opponents (aprox half way down on right image) work using ratios, we can trick them for half the game and take advantage for the remainder. Others are much shorter term solution, alternating between C and D every other turn (1/4 down the left image). When scoring in the mid range (bottom of right, top of left) there seems to be patterns too; totalities of C are to be expected but there are some results that seem to be tricking an opponent for the first number of moved before defecting for the rest of the game. At the higher score ranges there seems to be lots of noise, many of these opponents are rather complicated in their Strategy, the top 12 non totalities are listed below:

One final point regarding what Figure 6.7 shows is the effectiveness of the genetic algorithm. From the theory of repeated games, an equilibria for the repeated game must end in an equilibria for the static game. Hence for one of our solution sequences to be optimal the result must end in a D. This is true for all but 57 of the solutions, meaning there is room for possible improvement in more than one result. These sub optimal results were all stochastic.

Cosine Distance

$$d(S_i, S_j) = cos(\theta) = \frac{S_i \cdot S_j}{||S_i||\ ||S_j||}$$

The cosine of two vectors constructed by using the dot product formula as shown above. In our interpretation each dimension represents a sequence element so we are working in $\mathbb{R}^{200}$ with every value taking a $C : 1$ a $D : 0$. Figure 6.4 shows the distance matrix generated from the data files using code in Figure B.3

The Matrix for Cosine and Hamming are almost exactly similar, the only difference being the value of the distance between sequences. This is due to the two measures being similar in their relationship of space [31]. We can conclude that there is no extra information shown in this diagram.

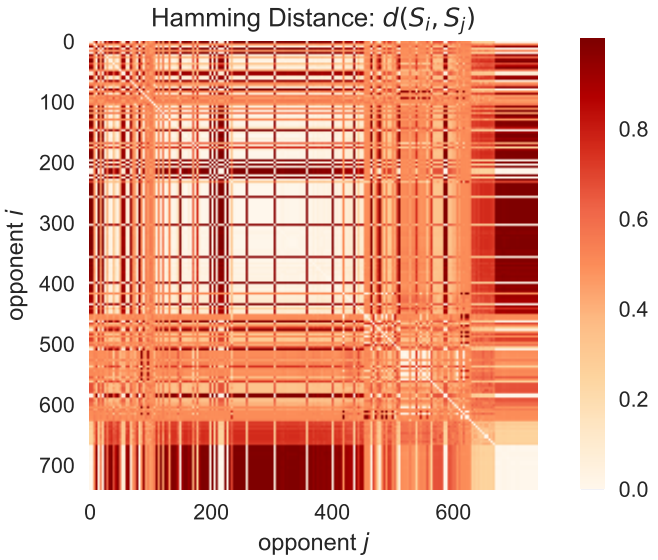

Figure 6.3: Distance Matrix for Hamming Distance

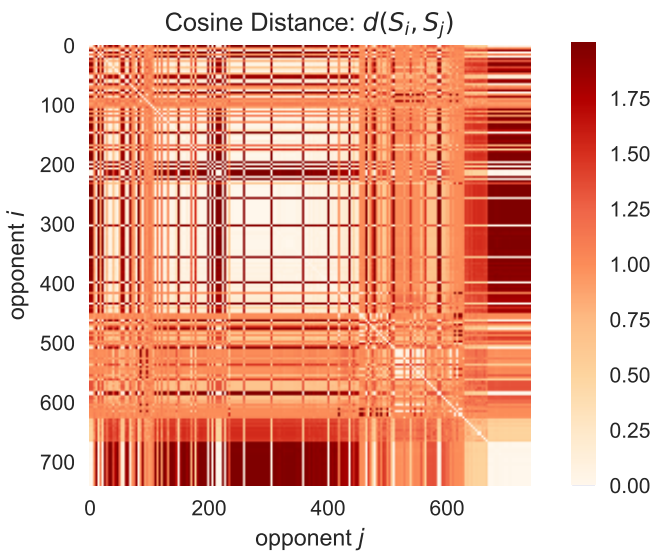

Figure 6.4: Distance Matrix for Cosine Distance

6.3 Solution Groups

If we want to group opponents together, the most obvious way is to look at which opponents have have the same best score sequence. Appendix section D has full details, but figure 6.5 shows a plot of the trends. We can observe that there are lots of sequences with solutions that dont contain many blocks (≤ 25) and the solutions get more varied in number fo blocks in the mid range of scores.

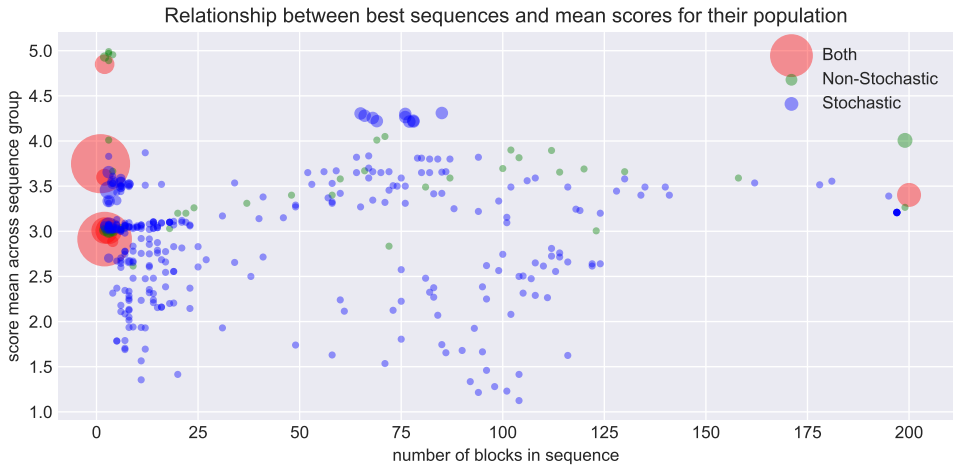

Figure 6.5: Trends for opponents grouped by their best sequence

This figure also shows an almost logarithmic/polynomial trend in the number of blocks in a sequence as we move up the score. Below this trend there are also a group of opponents who dont seem to follow the pattern.

6.4 Clustering Analysis using SciKitLearn

Here we will look at ways of grouping opponents and what that means for potential scoring outcomes. These clustering algorithms are not meant for predictive purposes as there is not enough metadata present in each strategy to use as features.

6.4.1 K Means clustering

Nothing much was identified from the k means clustering data analysis. This is probably due to the fact that the clustering was conducted using secondary, extrapolated data. Figure 6.8 shows the clustering for the most correlated variables in 3 dimensions. Its clear that these clusters are layered over the number of blocks of the solutions sequence. Section 6.3 looks in depth as to how opponents are distributed over the solution sequences.

6.4.2 Regression Trees

Regression trees are a way of looking at reducing the variance of a parameter through attributes of observed instances. ScikitLearn has an implementation of regression trees, the code used to generate the data in this subsection is shown in appendix code B.4.

Our case of regression trees we're looking at reducing the variance of the score by looking at which turns tend to cause the largest disparity in score. To do this we look at reducing the Mean Absolute Error of the scores, which is, in effect reducing the $L1$ loss of the scores.

$$MAE = \frac{1}{n}\sum_{i=0}^{n}|x_i - y_i|$$

Hence Figure 6.9 shows which moves cause the largest variance in scores. As you move right on the tree, or move in the false direction, then you're playing cooperations at the moves listed. The diagram backs up that by playing C moves consistently (furthest right leaf) we reach a score value of 3.01, and with 325 samples and $MAE = 0.205$ this is a clear way of doing well against a large number of opponents. If we move left on the diagram then the score value increases, which is to be expected; these are the best turns to defect and get a higher score as a result.

The move that dictates the best change in score is move 164 (starting from the 0th turn), if we defect on this move then the score value goes up but so does the MAE. Looking at leaf nodes provides an overview of solutions being grouped into minimal MAE after considering 5 moves. These represent which moves are played on the same turn by multiple solutions, all who scored approximately the same. Its clear that there are some paths that, considering one move difference, have clusters of very different scores. For example, the 3rd and 4th groups from the left (with 19 and 4 members respectively) show that, on the turn 156, you can defect and most likely get around a 3.2 or cooperate and get half that score. This shows that even though some solutions are similar, the corresponding opponents are vastly different in operation and will punish at some points others would forgive.

This tree doesn't have very many useful properties other than displaying the complexities in trying to predict a solution for any opponent. As discussed in Section 6.3 if an opponent doesn't sit in one of the major groups it becomes incredibly hard to observe some sort of pattern between them. One solution may be to look into combinations of initial handshakes that create the largest variation of responses in the population of opponents; from this we could map these results to a best response sequence to play for the remainder of the game.

6.5 Conclusion

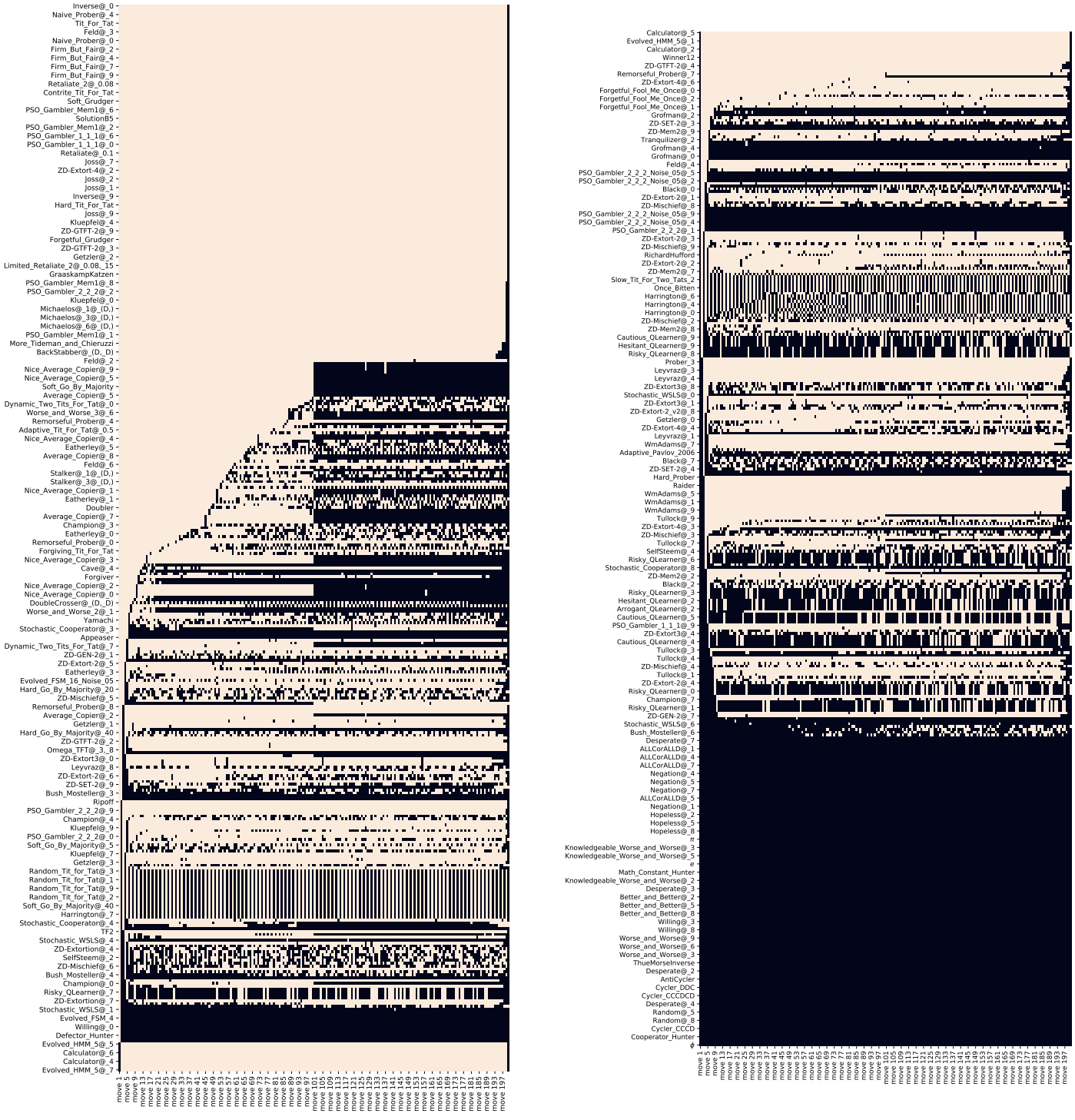

Figure 6.6: Sequence Diagram, sorted by move in turns. *note: labels are not complete, approx 1/4 shown.*

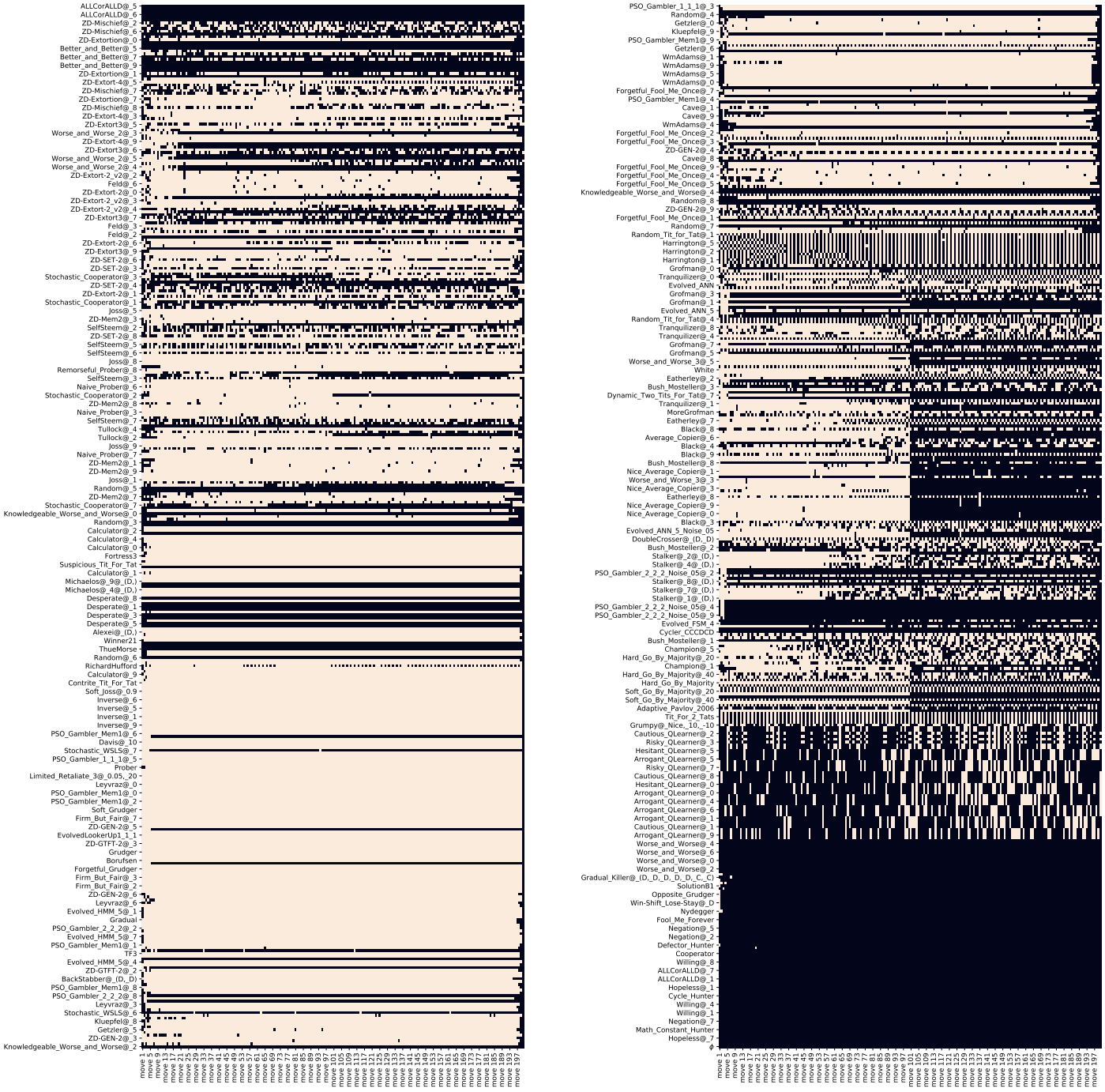

Figure 6.7: Sequence Diagram, sorted by score. *note: labels are not complete, approx 1/4 shown.*

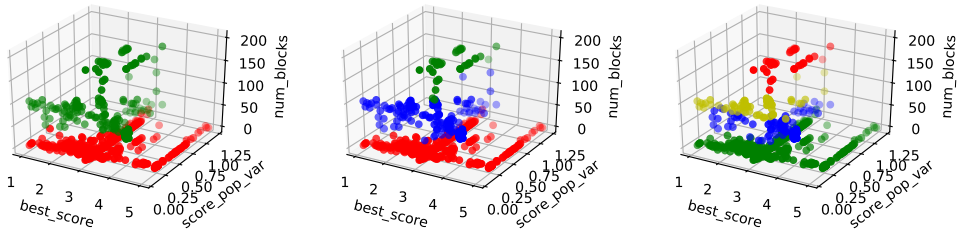

Figure 6.8: K means clustering with 2,3 and 4 clusters

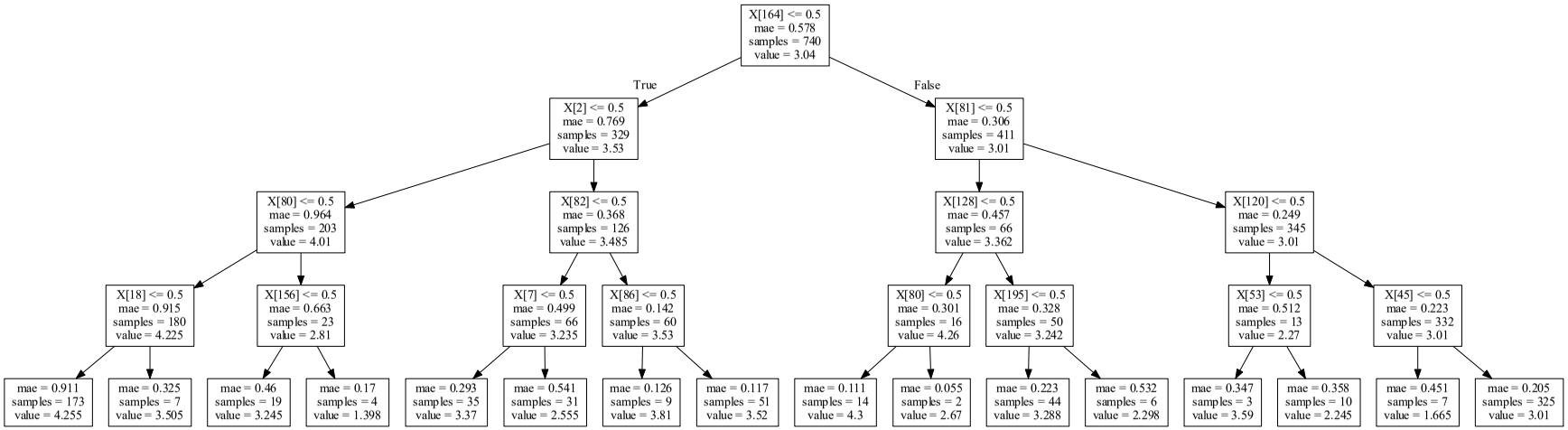

Figure 6.9: A regression tree showing which moves introduce the largest absolute error in the best score. If $X[i] <= 0.5$ is true, it means move i is a Defection move. **TRUE or left** $\Rightarrow D$, **FALSE or right** $\Rightarrow C$

Chapter 7

Practical Applications for Solution Sequences

maybe how this works with respect to pathfinding, who we should select in a set of opponents if we only have to play a single opponent in the set. If we can score an average of 3.8 against op1 or 3 against op2, we play op2 etc.

game shows to win money? We want to win as much money as possible but we don't mind how much our opponent wins, as long as we get lots.

What were actually doing is finding a sequence that will manipulate our opponent into providing us the most number of cooperation moves we can subsequently defect against. Failing that we are manipulating the opponent for cooperation moves we can cooperate against, then defection moves we can defect against, then defection moves we must cooperate for.

Chapter 8

Summary and Future Research

This part should defiantly include the fact it was hard to get information about the opponents. Classification of opponents was quite difficult.

Appendix A

Online Resources

- Github Code - https://github.com/GitToby/FinalYearProject

Appendix B

Code Appendix

```
def runGeneticAlgo(opponent, population_size=150, number_of_game_turns=200, cycle_length=200, generations=250,
                   mutation_probability=0.1, mutation_potency=1, reset_file=True):
    cycler_class = axl_dojo.CyclerParams
    cycler_objective = axl_dojo.prepare_objective(name="score", turns=number_of_game_turns, repetitions=1)
    cycler_kwargs = {
        "sequence_length": cycle_length,
        "mutation_probability":mutation_probability,
        "mutation_potency":mutation_potency
    }

    output_file_name = "data/" + str(opponent).replace(" ","_") + ".csv"
    try:
        if reset_file and os.path.isfile(output_file_name):
            os.remove(output_file_name)
    finally:
        print(str(opponent),
              "|| pop size:", population_size,
              "\tturns:", number_of_game_turns,
              "\tcycle len:", cycle_length,
              "\tgens:", generations,
              "\tmut. rate:",mutation_probability,
              "\t, potency:",mutation_potency)

        axl.seed(1)

        population = axl_dojo.Population(params_class=cycler_class,
                                         params_kwargs=cycler_kwargs,
                                         size=population_size,
                                         objective=cycler_objective,
                                         processes=0,
                                         output_filename=output_file_name,
                                         opponents=[opponent],
                                         print_output=False)
        population.run(generations)
        print("\tAnalysis Complete:",output_file_name)
    # Store the file name and opponent name as a tuple
    return output_file_name, str(opponent)
```

Figure B.1: Code for testing the genetic algorithm in Jupyter notebooks.

```
class NewAnalysisRun:
    # default options
    opponent_list = []
    output_files = {}
    save_directory = "output/"
    save_file_prefix = ""
    save_file_suffix = ""
    global_seed = 0
    overwrite_files = True
    stochastic_seeds = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

    def __init__(self, sequence_length=20,
                 population_size=25,
                 generation_length=20,
                 mutation_frequency=0.1,
                 mutation_potency=1):
        self.sequence_length = sequence_length
        self.population_size = population_size
        self.generation_length = generation_length
        self.mutation_frequency = mutation_frequency
        self.mutation_potency = mutation_potency

    def get_pre_made_pop(self, pop_size: int):
        pop = []

        # Totalities & Handshakes
        handshake_leng = 5
        for start in itertools.product("CD", repeat=handshake_leng):
            pop.append(axl_dojo.CyclerParams(
                list(start) + [C] * (200 - handshake_leng)))
            pop.append(axl_dojo.CyclerParams(
                list(start) + [D] * (200 - handshake_leng)))

        # 50-50
        pop.append(axl_dojo.CyclerParams([C] * 100 + [D] * 100))
        pop.append(axl_dojo.CyclerParams([D] * 100 + [C] * 100))

        # Single Change
        for i in range(1, 11):
            pop.append(axl_dojo.CyclerParams([C] * i + [D] * (200 - i)))
            pop.append(axl_dojo.CyclerParams([D] * i + [C] * (200 - i)))

        for i in range(1, 11):
            pop.append(axl_dojo.CyclerParams([C] * (200 - i) + [D] * i))
            pop.append(axl_dojo.CyclerParams([D] * (200 - i) + [C] * i))

        # Matching Tails
        for i in range(1, 6):
            for j in range(1, 6):
                pop.append(axl_dojo.CyclerParams(
                    [C] * i + [D] * (200 - (i + j)) + [C] * j))
                pop.append(axl_dojo.CyclerParams(
                    [D] * i + [C] * (200 - (i + j)) + [D] * j))

        # Alternating
        pop.append(axl_dojo.CyclerParams([C, D] * 100))
        pop.append(axl_dojo.CyclerParams([D, C] * 100))
        pop.append(axl_dojo.CyclerParams([C, C, D, D] * 50))
        pop.append(axl_dojo.CyclerParams([D, D, C, C] * 50))
        pop.append(axl_dojo.CyclerParams([C, C, C, C, D, D, D, D] * 25))
        pop.append(axl_dojo.CyclerParams([D, D, D, D, C, C, C, C] * 25))
        pop.append(axl_dojo.CyclerParams([C, C, C, C, C, D, D, D, D, D] * 20))
        pop.append(axl_dojo.CyclerParams([D, D, D, D, D, C, C, C, C, C] * 20))

        # Random Filler
        while len(pop) < pop_size:
            random_moves = list(map(axl.Action, np.random.randint(
                0, 1 + 1, (self.sequence_length, 1))))
            pop.append(axl_dojo.CyclerParams(random_moves))
```

```
from sklearn import metrics


def CD_map_to_int(x):
    # Returns D:0 C:1
    return 68 - ord(x)


# Itertuples is a list of (index,col1,col2,..) for each row. We sort them by score first
df_vectors = pd.DataFrame()
df_vectors_strings = pd.DataFrame()
df_generation_max = df_generation_max.sort_values('opponent_name')
df_generation_max = df_generation_max.sort_values('best_score')
for tup in df_generation_max.itertuples():
    # index vals mean the indexing starts at 1 so +1 to these:
    #(0, 'generation'), (1, 'score_mean'), (2, 'score_median'), (3, 'score_pop_var'), (4, 'score_range'), (5, 'best_score
    seq_str = tup[7]
    opponent_name = str(tup[8])
    best_score = tup[6]
    bins = tup[15]
    # Mapping to integers 0 &1
    df_vectors[opponent_name] = list(
        map(CD_map_to_int, seq_str)) + [best_score] + [seq_str] + [bins]

    # No Mapping Cs & Ds
    df_vectors_strings[opponent_name] = list(
        seq_str) + [best_score] + [seq_str] + [bins]


move_cols = ['move ' + str(x+1) for x in range(200)]
# Transposing and labeling moves as were forming it in a rotated way.
# (Its easier to just make 2 dfs than map one to the other)
df_vectors = df_vectors.transpose()
df_vectors.columns = move_cols + ["best_score", "best_sequence", "score_bins"]

df_vectors_strings = df_vectors_strings.transpose()
df_vectors_strings.columns = move_cols + \
    ["best_score", "best_sequence", "score_bins"]

# Manhatten Distance == Hamming Distance in this example
# Intresting examples: cosine(??), hamming, jaccard sim
dist_array_ham = metrics.pairwise.pairwise_distances(
    df_vectors[move_cols], metric='hamming')
# add labels
dist_array_ham = pd.DataFrame(
    dist_array_ham, index=df_vectors.index, columns=df_vectors.index)

dist_array_cos = metrics.pairwise.pairwise_distances(
    df_vectors[move_cols], metric='cosine')
dist_array_cos = pd.DataFrame(
    dist_array_cos, index=df_vectors.index, columns=df_vectors.index)

dist_array_jac = metrics.pairwise.pairwise_distances(
    df_vectors[move_cols], metric='jaccard')
dist_array_cos = pd.DataFrame(
    dist_array_cos, index=df_vectors.index, columns=df_vectors.index)

dist_array_other = metrics.pairwise.pairwise_distances(
    df_vectors[move_cols], metric='correlation')
dist_array_cos = pd.DataFrame(
    dist_array_cos, index=df_vectors.index, columns=df_vectors.index)
```

Figure B.3: Distance Matrix generation code using SciKitLearn

```
# We use mean absolut error, as we want to penalise the magnitude of the error linearly
# (rmse would how how the variance in our scores is reduced move by move)
regressor_tree = tree.DecisionTreeRegressor(criterion='mae',max_depth=4)
#Rerun this for # of cs in a sequence.
regressor_tree.fit(df_vectors[move_cols],df_vectors["best_score"],)

# This will export the graph to a .dot file, use `dot -Tpng .\tree.dot -o tree.png` in cmd to convert to png
# dot_data_clasif = tree.export_graphviz(regressor_tree, out_file='tree.dot')
dot_data_reg = tree.export_graphviz(regressor_tree, out_file='reg_tree.dot')
# Run on CMD line
!dot -Tpng .\reg_tree.dot -o reg_tree.png
!dot -Tpdf .\reg_tree.dot -o reg_tree.pdf

i = Image.open('./reg_tree.png')
plt.figure(figsize=(20,20))
plt.imshow(i)
```

Figure B.4: Regression trees using SciKitLearn

Appendix C

List of Axelrod Opponents

* means Stochastic Opponent, (LRT) means Long Run Time.

- ϕ
- π
- e
- ALLCorALLD*
- Adaptive
- Adaptive Pavlov 2006
- Adaptive Pavlov 2011
- Adaptive Tit For Tat
- Aggravater
- Alexei
- Alternator
- Alternator Hunter
- Anti Tit For Tat
- AntiCycler
- Appeaser
- Arrogant QLearner*
- Average Copier*
- BackStabber
- Better and Better*
- Black*
- Borufsen
- Bully
- Bush Mosteller*
- Calculator*
- Cautious QLearner*
- Cave*
- Champion*
- Colbert
- CollectiveStrategy
- Contrite Tit For Tat
- Cooperator
- Cooperator Hunter
- Cycle Hunter
- Cycler CCCCCD
- Cycler CCCD
- Cycler CCCDCD
- Cycler CCD
- Cycler DC
- Cycler DDC
- DBS (LRT)
- Davis
- Defector
- Defector Hunter
- Desperate*
- DoubleCrosser
- DoubleResurrection
- Doubler
- Dynamic Two Tits For Tat*
- EasyGo
- Eatherley*
- EugineNier
- Eventual Cycle Hunter
- Evolved ANN
- Evolved ANN 5
- Evolved ANN 5 Noise 05
- Evolved FSM 16
- Evolved FSM 16 Noise 05
- Evolved FSM 4
- Evolved HMM 5*
- EvolvedLookerUp1_1_1
- EvolvedLookerUp2_2_2
- Feld*
- Firm But Fair*
- Fool Me Forever
- Fool Me Once
- Forgetful Fool Me Once*
- Forgetful Grudger
- Forgiver
- Forgiving Tit For Tat
- Fortress3
- Fortress4
- GTFT*
- General Soft Grudger
- Getzler*
- Gladstein
- Go By Majority
- Go By Majority 10
- Go By Majority 20
- Go By Majority 40
- Go By Majority 5
- GraaskampKatzen
- Gradual
- Gradual Killer
- Grofman*
- Grudger
- GrudgerAlternator
- Grumpy
- Handshake
- Hard Go By Majority
- Hard Go By Majority 10

- Hard Go By Majority 20
- Hard Go By Majority 40
- Hard Go By Majority 5
- Hard Prober
- Hard Tit For 2 Tats
- Hard Tit For Tat
- Harrington*
- Hesitant QLearner*
- Hopeless*
- Inverse*
- Inverse Punisher
- Joss*
- Kluepfel*
- Knowledgeable Worse and Worse*
- Level Punisher
- Leyvraz*
- Limited Retaliate
- Limited Retaliate 2
- Limited Retaliate 3
- MEM2
- Math Constant Hunter
- Meta Hunter
- Meta Hunter Aggressive
- Meta Majority* (LRT)
- Meta Majority Finite Memory* (LRT)
- Meta Majority Long Memory* (LRT)
- Meta Majority Memory One* (LRT)
- Meta Minority* (LRT)
- Meta Mixer* (LRT)
- Meta Winner* (LRT)
- Meta Winner Deterministic* (LRT)
- Meta Winner Ensemble* (LRT)
- Meta Winner Finite Memory* (LRT)
- Meta Winner Long Memory* (LRT)
- Meta Winner Memory One* (LRT)
- Meta Winner Stochastic* (LRT)
- Michaelos*
- More Tideman and Chieruzzi
- MoreGrofman
- N Tit(s) For M Tat(s)
- NMWE Deterministic* (LRT)
- NMWE Finite Memory* (LRT)
- NMWE Long Memory* (LRT)
- NMWE Memory One* (LRT)
- NMWE Stochastic* (LRT)
- Naive Prober*
- Negation*
- Nice Average Copier*
- Nice Meta Winner* (LRT)
- Nice Meta Winner Ensemble* (LRT)
- Nydegger
- Omega TFT
- Once Bitten
- Opposite Grudger
- PSO Gambler 1_1_1*
- PSO Gambler 2_2_2*
- PSO Gambler 2_2_2 Noise 05*
- PSO Gambler Mem1*
- Predator
- Prober
- Prober 2
- Prober 3
- Prober 4
- Pun1
- Punisher
- Raider
- Random*
- Random Hunter
- Random Tit for Tat*
- Remorseful Prober*
- Resurrection
- Retaliate
- Retaliate 2
- Retaliate 3
- Revised Downing
- RichardHufford
- Ripoff
- Risky QLearner*
- SelfSteem*
- ShortMem
- Shubik
- Slow Tit For Two Tats 2
- Sneaky Tit For Tat
- Soft Grudger
- Soft Joss*
- SolutionB1
- SolutionB5
- Spiteful Tit For Tat
- Stalker*
- Stein and Rapoport
- Stochastic Cooperator*
- Stochastic WSLS*
- Suspicious Tit For Tat
- TF1
- TF2
- TF3
- Tester
- ThueMorse
- ThueMorseInverse
- Thumper
- Tideman and Chieruzzi
- Tit For 2 Tats
- Tit For Tat
- Tranquilizer*
- Tricky Cooperator
- Tricky Defector
- Tullock*
- Two Tits For Tat
- VeryBad
- Weiner
- White
- Willing*
- Win-Shift Lose-Stay
- Win-Stay Lose-Shift
- Winner12
- Winner21
- WmAdams*
- Worse and Worse*
- Worse and Worse 2*
- Worse and Worse 3*
- Yamachi
- ZD-Extort-2*
- ZD-Extort-2 v2*
- ZD-Extort-4*
- ZD-Extort3*
- ZD-Extortion*
- ZD-GEN-2*
- ZD-GTFT-2*
- ZD-Mem2*
- ZD-Mischief*
- ZD-SET-2*

Appendix D

List of Best Solution Sequences

- C199,1 for 96 opponents:
  - Aggravater (score:2.965)
  - Borufsen (score:3.01)
  - Cave@_5 (score:3.01)
  - Contrite_Tit_For_Tat (score:3.01)
  - Davis@_10 (score:3.01)
  - EvolvedLookerUp1_1_1 (score:3.01)
  - Feld@_3 (score:2.29)
  - Firm_But_Fair@_0 (score:3.01)
  - Firm_But_Fair@_1 (score:3.01)
  - Firm_But_Fair@_2 (score:3.01)
  - Firm_But_Fair@_3 (score:3.01)
  - Firm_But_Fair@_4 (score:3.01)
  - Firm_But_Fair@_5 (score:3.01)
  - Firm_But_Fair@_6 (score:3.01)
  - Firm_But_Fair@_7 (score:3.01)
  - Firm_But_Fair@_8 (score:3.01)
  - Firm_But_Fair@_9 (score:3.01)
  - Forgetful_Grudger (score:3.01)
  - General_Soft_Grudger@_n=1,d=4,c=2 (score:3.01)
  - Getzler@_2 (score:3.01)
  - GraaskampKatzen (score:3.01)
  - Grudger (score:3.01)
  - Hard_Tit_For_Tat (score:3.01)
  - Inverse@_0 (score:3.01)
  - Inverse@_1 (score:3.01)
  - Inverse@_2 (score:3.01)
  - Inverse@_3 (score:3.01)
  - Inverse@_4 (score:3.01)
  - Inverse@_5 (score:3.01)
  - Inverse@_6 (score:3.01)
  - Inverse@_7 (score:3.01)
  - Inverse@_8 (score:3.01)
  - Inverse@_9 (score:3.01)
  - Inverse_Punisher (score:3.01)
  - Joss@_0 (score:2.665)
  - Joss@_1 (score:2.8)
  - Joss@_2 (score:2.65)
  - Joss@_3 (score:2.695)
  - Joss@_4 (score:2.635)
  - Joss@_5 (score:2.59)
  - Joss@_7 (score:2.77)
  - Joss@_8 (score:2.65)
  - Joss@_9 (score:2.755)
  - Kluepfel@_1 (score:3.01)
  - Kluepfel@_4 (score:3.01)
  - Leyvraz@_0 (score:3.01)
  - Limited_Retaliate@_0.1,_20 (score:3.01)
  - Limited_Retaliate_2@_0.08,_15 (score:3.01)
  - Limited_Retaliate_3@_0.05,_20 (score:3.01)
  - MEM2 (score:3.01)
  - Meta_Hunter@_6_players (score:3.01)
  - Naive_Prober@_0 (score:2.665)
  - Naive_Prober@_1 (score:2.8)
  - Naive_Prober@_2 (score:2.65)
  - Naive_Prober@_3 (score:2.695)
  - Naive_Prober@_4 (score:2.635)
  - Naive_Prober@_5 (score:2.59)
  - Naive_Prober@_7 (score:2.77)
  - Naive_Prober@_8 (score:2.65)
  - Naive_Prober@_9 (score:2.755)
  - PSO_Gambler_1_1_1@_0 (score:3.01)
  - PSO_Gambler_1_1_1@_2 (score:3.01)
  - PSO_Gambler_1_1_1@_5 (score:3.01)
  - PSO_Gambler_1_1_1@_6 (score:3.01)
  - PSO_Gambler_Mem1@_0 (score:3.01)
  - PSO_Gambler_Mem1@_2 (score:3.01)
  - PSO_Gambler_Mem1@_5 (score:3.01)
  - PSO_Gambler_Mem1@_6 (score:3.01)
  - Punisher (score:3.01)
  - Remorseful_Prober@_1 (score:2.8)
  - Remorseful_Prober@_9 (score:2.755)
  - Resurrection (score:3.01)

- Retaliate@_0.1 (score:3.01)
- Retaliate_2@_0.08 (score:3.01)
- Retaliate_3@_0.05 (score:3.01)
- Shubik (score:3.01)
- Soft_Grudger (score:3.01)
- Soft_Joss@_0.9 (score:3.01)
- SolutionB5 (score:2.995)
- Spiteful_Tit_For_Tat (score:3.01)
- Suspicious_Tit_For_Tat (score:2.995)
- Thumper (score:3.01)
- Tideman_and_Chieruzzi (score:3.01)
- Tit_For_Tat (score:3.01)
- Two_Tits_For_Tat (score:3.01)
- ZD-Extort-4@_2 (score:1.825)
- ZD-Extort-4@_9 (score:1.945)
- ZD-GEN-2@_2 (score:3.01)
- ZD-GEN-2@_5 (score:3.01)
- ZD-GTFT-2@_0 (score:3.01)
- ZD-GTFT-2@_1 (score:3.01)
- ZD-GTFT-2@_3 (score:3.01)
- ZD-GTFT-2@_5 (score:3.01)
- ZD-GTFT-2@_6 (score:3.01)
- ZD-GTFT-2@_8 (score:3.01)
- ZD-GTFT-2@_9 (score:3.01)

- C198,2 for 21 opponents:
  - Alexei@_(D,) (score:3.0)
  - EugineNier@_(D,) (score:3.0)
  - Fool_Me_Once (score:3.02)
  - Kluepfel@_0 (score:3.02)
  - Kluepfel@_5 (score:3.02)
  - Michaelos@_0@_(D,) (score:3.0)
  - Michaelos@_1@_(D,) (score:3.0)
  - Michaelos@_2@_(D,) (score:3.0)
  - Michaelos@_3@_(D,) (score:3.0)
  - Michaelos@_4@_(D,) (score:3.0)
  - Michaelos@_5@_(D,) (score:3.0)
  - Michaelos@_6@_(D,) (score:3.0)
  - Michaelos@_7@_(D,) (score:3.0)
  - Michaelos@_8@_(D,) (score:3.0)
  - Michaelos@_9@_(D,) (score:3.0)
  - PSO_Gambler_2_2_2@_2 (score:3.02)
  - PSO_Gambler_2_2_2@_6 (score:3.02)
  - PSO_Gambler_Mem1@_1 (score:3.02)
  - PSO_Gambler_Mem1@_3 (score:3.02)
  - PSO_Gambler_Mem1@_8 (score:3.02)
  - TF3 (score:3.02)
- C196,4 for 3 opponents:
  - Gradual (score:3.02)
  - More_Tideman_and_Chieruzzi (score:3.02)
  - ZD-GEN-2@_3 (score:3.04)
- C194,6 for 1 opponents:
  - BackStabber@_(D,_D) (score:3.02)
- C193,7 for 2 opponents:
  - PSO_Gambler_Mem1@_4 (score:3.07)
  - PSO_Gambler_Mem1@_9 (score:3.05)
- C151,1,47,1 for 1 opponents:
  - Feld@_2 (score:2.315)
- C100,26,1,9,1,63 for 3 opponents:
  - Dynamic_Two_Tits_For_Tat@_9 (score:3.545)
  - Nice_Average_Copier@_9 (score:3.545)
  - Worse_and_Worse_3@_9 (score:3.545)
- C100,36,1,63 for 1 opponents:

  - Average_Copier@_9 (score:3.535)

- C100,100 for 8 opponents:
  - Average_Copier@_5 (score:3.365)
  - Dynamic_Two_Tits_For_Tat@_3 (score:3.3)
  - Dynamic_Two_Tits_For_Tat@_5 (score:3.38)
  - Hard_Go_By_Majority (score:3.985)
  - Nice_Average_Copier@_5 (score:3.38)
  - Soft_Go_By_Majority (score:4.0)
  - VeryBad (score:4.0)
  - Worse_and_Worse_3@_5 (score:3.38)

- C99,1,3,1,1,1,2,2,3,1,2,1,1,2,10,2,1,1,1,1,2,1,3,1,1,1,3,1,3,1,1,1,3,1,3,5,1,2,1,1,2,1,3,1,1,1,1,3,1,1,1,1,4,1,2,1,1,2 for 1 opponents:
  - Level_Punisher (score:3.4)

- C98,2,1,4,1,9,2,1,3,2,2,2,1,1,1,1,1,1,2,4,1,1,1,2,1,1,2,4,1,5,1,1,2,4,4,5,2,1,6,4,2,1,1,1,1,1,3,1,1 for 1 opponents:
  - Dynamic_Two_Tits_For_Tat@_2 (score:3.29)

- C95,3,1,1,1,3,1,3,1,2,1,1,3,1,5,1,1,1,7,3,6,1,1,3,3,2,2,6,1,2,1,1,1,1,1,4,6,5,1,3,4,1,1,1,2,1,1,2,1 for 1 opponents:
  - Dynamic_Two_Tits_For_Tat@_0 (score:3.27)

- C92,3,3,1,2,1,2,1,2,1,11,1,1,4,7,1,5,1,1,3,4,1,1,2,4,1,1,1,1,1,1,1,1,2,2,1,2,2,3,2,4,1,4,1,5,6,1,1 for 1 opponents:
  - White (score:3.4)

- C87,1,1,3,1,1,2,2,1,1,1,3,1,3,2,1,2,2,4,1,1,1,3,3,2,1,4,1,1,3,2,2,2,2,2,4,5,1,2,1,2,4,1,3,3,1,1,2,1,2,1,2,3,1,5,1,1,1,1,1 for 1 opponents:
  - Feld@_8 (score:2.24)

- C87,3,2,2,6,12,1,87 for 2 opponents:
  - Nice_Average_Copier@_6 (score:3.52)
  - Worse_and_Worse_3@_6 (score:3.52)

- C87,3,2,2,6,100 for 1 opponents:
  - Average_Copier@_6 (score:3.495)

- C86,1,93,1,18,1 for 1 opponents:
  - Remorseful_Prober@_4 (score:2.645)

- C85,1,12,1,1,74,1,25 for 1 opponents:
  - Dynamic_Two_Tits_For_Tat@_8 (score:3.495)

- C84,1,5,1,1,1,1,3,1,3,1,1,1,3,3,2,1,1,1,2,2,1,1,1,1,7,1,6,1,2,1,4,1,1,4,2,2,4,5,6,1,2,3,1,3,1,1,4,6,5,1,2 for 1 opponents:

- Stalker@_5@_(D,) (score:3.65)

- C79,1,54,1,63,2 for 1 opponents:
  - Adaptive_Tit_For_Tat@_0.5 (score:3.04)

- C78,1,1,1,1,3,1,2,4,1,6,2,3,1,1,1,1,3,1,1,2,2,1,1,2,1,4,1,1,1,2,3,3,1,5,1,1,2,5,1,1,2,1,2,2,7,2,2,2,4,3,1,3,5,2,1,2,1,1,2,1 for 1 opponents:
  - Feld@_5 (score:2.115)

- C71,1,28,26,1,73 for 2 opponents:
  - Nice_Average_Copier@_4 (score:3.585)
  - Worse_and_Worse_3@_4 (score:3.585)

- C71,1,17,9,1,6,6,1,1,1,2,2,9,2,4,1,1,1,3,1,2,1,2,1,2,10,3,3,2,3,6,1,2,3,3,2,1,10,1,1,1 for 1 opponents:
  - Dynamic_Two_Tits_For_Tat@_1 (score:3.38)

- C71,3,1,2,7,3,2,3,5,1,3,6,6,4,1,1,3,1,1,1,1,1,3,4,1,4,2,6,1,1,4,1,2,1,2,1,2,2,1,1,2,4,2,2,1,1,2,1,1,1,1,2,3,5,1,5 for 1 opponents:
  - Stalker@_7@_(D,) (score:3.66)

- C67,1,1,1,1,1,1,1,1,1,1,1,1,3,1,1,1,1,1,1,1,3,1,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
for 1 opponents:
  - Eatherley@_5 (score:3.445)

- C66,1,1,2,2,1,7,1,5,2,2,1,2,1,1,1,1,2,1,1,1,2,1,5,2,1,4,7,2,3,2,1,3,2,2,1,1,1,2,3,2,1,2,2,1,4,1,1,1,4,2,1,3,2,2,3,3,2,6,3,1,3,1,2 for 1 opponents:
  - Stalker@_6@_(D,) (score:3.65)

- C65,3,1,2,3,2,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
for 1 opponents:
  - Tranquilizer@_6 (score:3.2)

- C64,2,34,100 for 1 opponents:
  - Average_Copier@_8 (score:3.53)

- C64,3,33,100 for 1 opponents:
  - Average_Copier@_4 (score:3.61)

- C62,1,2,2,2,1,3,1,7,1,9,1,1,2,3,4,5,5,5,2,1,3,4,1,2,1,6,3,2,1,1,2,2,1,3,1,1,2,3,2,1,1,1,1,4,6,3,1,1,3,3,1,6,2,1,2,2 for 1 opponents:

- Tranquilizer@_9 (score:3.37)

- C59,1,116,1,21,2 for 1 opponents:
  - Feld@_6 (score:2.18)

- C58,1,1,1,1,1,1,2,1,2,2,2,2,2,5,5,1,1,1,3,1,2,2,3,2,2,3,1,1,2,1,3,1,2,5,1,2,3,2,2,2,1,2,1,2,2,1,1,6,1,4,3,5,1,3,1,3,2,1,1,2,1,1,3,5,1,2,3,1
  for 1 opponents:
  - Stalker@_2@_(D,) (score:3.65)

- C57,1,42,8,1,44,1,46 for 1 opponents:
  - Average_Copier@_1 (score:3.52)

- C56,5,4,1,3,2,5,1,2,1,1,1,1,1,1,2,3,1,2,7,2,1,2,1,2,8,2,1,1,1,1,1,1,1,5,2,1,1,2,1,5,1,1,2,1,3,4,2,2,3,5,3,2,1,2,1,5,2,2,2,2,1,1,3,2,1,1,3
  for 1 opponents:
  - Stalker@_1@_(D,) (score:3.66)

- C55,1,2,1,1,1,1,1,1,4,1,1,1,3,2,1,5,1,3,1,2,2,7,1,1,1,1,4,1,1,2,2,2,4,1,1,1,2,2,3,2,1,4,1,1,3,2,2,1,1,1,1,1,1,2,3,7,1,1,2,1,3,2,1,3,3,2,1,2
  for 1 opponents:
  - Stalker@_4@_(D,) (score:3.65)

- C55,2,2,1,2,1,1,1,1,1,10,1,22,14,1,85 for 1 opponents:
  - Dynamic_Two_Tits_For_Tat@_6 (score:3.52)

- C53,1,3,1,1,2,2,1,1,3,1,5,2,3,3,3,2,1,1,2,1,2,5,1,1,3,1,2,2,1,4,1,4,1,1,5,1,1,1,1,4,2,11,2,4,1,1,1,4,3,1,2,1,2,1,3,2,1,1,1,2,1,1,1,2,1,2,2,4
  for 1 opponents:
  - Stalker@_3@_(D,) (score:3.66)

- C52,1,32,1,113,1 for 1 opponents:
  - Remorseful_Prober@_3 (score:2.705)

- C52,1,2,1,5,1,2,2,1,2,3,1,1,1,3,1,2,1,2,3,1,2,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
  for 1 opponents:
  - Eatherley@_9 (score:3.58)

- C50,1,49,33,1,7,1,58 for 2 opponents:
  - Nice_Average_Copier@_1 (score:3.525)
  - Worse_and_Worse_3@_1 (score:3.525)

- C50,1,14,2,1,1,4,3,3,1,6,2,2,1,2,1,1,2,3,4,1,1,1,3,1,1,1,2,1,1,1,1,1,2,3,1,1,1,1,1,4,7,1,2,2,4,1,2,1,3,1,1,1,1,1,4,3,2,2,3,1,2,3,2,1,1,1,1,2
  for 1 opponents:

- Eatherley@_6 (score:3.5)

- C50,1,1,1,7,2,4,1,1,1,1,2,1,1,2,1,1,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
for 1 opponents:
  - Eatherley@_1 (score:3.49)

- C50,1,1,1,1,1,1,1,1,1,1,2,30,1,4,3,1,2,2,1,3,2,4,3,1,1,4,1,2,7,5,1,1,2,1,2,1,1,1,1,2,2,1,1,2,1,1,2,1,1,1,1,2,2,1,3,7,1,1,1,2,1,2,2,6,1,1,1
for 1 opponents:
  - Tranquilizer@_4 (score:3.345)

- C48,1,15,1,4,1,1,1,11,1,10,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
for 1 opponents:
  - ZD-Extort-4@_5 (score:1.625)

- C47,2,1,3,2,2,2,1,27,3,1,3,2,4,1,1,2,1,1,3,6,1,1,3,1,1,3,3,1,3,3,1,1,1,4,2,2,1,1,1,4,1,1,7,1,1,2,1,1,4,2,1,3,2,3,1,1,2,1,1,2,1,1,3,2,2
for 1 opponents:
  - Doubler (score:3.67)

- C46,1,51,1,1,100 for 2 opponents:
  - Nice_Average_Copier@_8 (score:3.49)
  - Worse_and_Worse_3@_8 (score:3.49)

- C44,1,55,100 for 3 opponents:
  - Average_Copier@_7 (score:3.55)
  - Nice_Average_Copier@_7 (score:3.55)
  - Worse_and_Worse_3@_7 (score:3.55)

- C44,1,2,1,2,3,1,1,1,2,1,2,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
for 1 opponents:
  - Champion@_3 (score:3.8)

- C42,4,22,1,5,1,5,2,2,1,5,1,1,1,1,2,3,1,2,3,3,1,3,4,3,1,1,1,6,3,1,1,2,1,1,1,4,1,1,1,2,1,1,1,2,1,5,2,4,1,1,2,2,2,3,1,2,1,1,1,2,1,1,1,6,1,1,1,
for 1 opponents:
  - Tranquilizer@_1 (score:3.455)

- C37,4,24,5,1,4,2,2,1,2,2,7,2,3,1,1,6,2,3,1,1,1,1,1,3,1,2,1,2,1,1,2,1,1,1,2,1,1,2,1,1,1,1,4,3,4,2,1,3,6,2,2,2,1,1,1,3,2,1,1,2,7,3,9
for 1 opponents:
  - Champion@_9 (score:3.82)

- C35,1,1,1,1,2,3,3,19,1,1,1,2,2,3,1,3,1,1,2,1,2,3,1,1,4,2,1,2,1,1,1,1,1,1,1,1,1,1,2,1,1,1,2,1,1,6,1,3,1,4,2,4,1,1,7,2,4,1,1,2,3,1,1,1,1,2,1,
for 1 opponents:
  - Eatherley@_0 (score:3.42)

- C31,1,1,3,1,2,2,1,2,1,1,2,3,1,1,2,2,2,2,1,1,3,3,2,2,2,1,1,2,3,1,2,1,1,4,1,2,1,1,1,6,1,2,1,4,1,4,5,3,2,1,1,1,1,2,4,5,5,4,1,1,2,4,1,4,3,1,1,1,
for 1 opponents:

- Champion@_5 (score:3.8)

- C31,2,40,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,5,3,1,4,3,2,1,2,1,5,1,3,1,1,1,2,2,5,1,1,1,4,1,6,2,2,1,1,1,1,6,1,3,5,1,1,1,3,1,
for 1 opponents:
  - Tranquilizer@_5 (score:3.31)

- C29,1,147,1,21,1 for 1 opponents:
  - Remorseful_Prober@_0 (score:2.675)

- C25,1,23,1,2,1,2,2,1,1,4,1,2,6,1,1,1,1,3,1,1,1,1,1,1,3,2,1,2,1,1,1,1,2,1,1,1,2,1,5,2,1,4,7,2,3,2,1,3,2,2,1,1,1,2,3,2,1,2,2,1,4,1,1,1,4,2,1,
for 1 opponents:
  - Champion@_6 (score:3.81)

- C22,1,24,4,4,4,1,1,2,1,1,4,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
for 1 opponents:
  - Eatherley@_2 (score:3.4)

- C21,1,28,1,2,1,4,1,15,1,10,1,29,1,5,1,6,1,52,1,8,10 for 1 opponents:
  - Forgiving_Tit_For_Tat (score:3.2)

- C16,1,41,1,2,1,1,1,3,1,1,2,1,2,5,1,1,5,1,2,1,1,4,2,1,2,1,1,2,1,2,1,2,2,3,1,2,1,2,1,9,2,2,2,1,1,3,1,1,2,1,2,1,1,1,1,1,1,1,1,2,3,1,1,7,2,2,2,
for 1 opponents:
  - Stalker@_0@_(D,) (score:3.65)

- C14,1,85,100 for 3 opponents:
  - Average_Copier@_3 (score:3.53)
  - Nice_Average_Copier@_3 (score:3.53)
  - Worse_and_Worse_3@_3 (score:3.53)

- C13,1,1,1,1,1,1,181 for 1 opponents:
  - Worse_and_Worse_2@_8 (score:2.255)

- C12,3,2,2,4,2,4,1,2,1,4,1,1,1,1,1,154,4 for 1 opponents:
  - Cave@_4 (score:3.1)

- C11,1,2,1,1,2,1,181 for 1 opponents:
  - Worse_and_Worse_2@_4 (score:2.125)

- C11,3,2,1,1,2,2,3,3,3,2,2,11,2,1,1,1,5,2,1,1,2,1,1,1,8,1,40,1,14,1,31,2,37 for 1 opponents:
  - Stochastic_Cooperator@_6 (score:2.655)

- C10,1,21,1,25,1,20,1,2,1,33,1,4,1,7,1,56,2,2,10 for 1 opponents:

- Forgiver (score:3.2)

- C10,1,2,1,2,1,2,181 for 1 opponents:
  - Worse_and_Worse_2@_3 (score:1.935)

- C10,3,4,1,1,2,1,178 for 1 opponents:
  - Worse_and_Worse_2@_6 (score:2.05)

- C9,1,90,26,1,73 for 2 opponents:
  - Nice_Average_Copier@_2 (score:3.485)
  - Worse_and_Worse_3@_2 (score:3.485)

- C9,1,90,57,1,42 for 2 opponents:
  - Nice_Average_Copier@_0 (score:3.55)
  - Worse_and_Worse_3@_0 (score:3.55)

- C9,1,2,1,2,1,2,182 for 1 opponents:
  - Worse_and_Worse_2@_7 (score:2.285)

- C8,1,5,2,1,183 for 1 opponents:
  - Worse_and_Worse_2@_5 (score:2.11)

- C7,1,1,1,1,1,1,1,1,1,2,1,2,1,1,1,2,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,2,1,1,1,2,1,1,1,1,1,1,2,1,1,1,1,1,2,1,1,1,1,1,1,1 for 1 opponents:
  - DoubleCrosser@_(D,_D) (score:3.59)

- C6,1,18,1,39,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, for 1 opponents:
  - Eatherley@_7 (score:3.49)

- C6,1,6,1,2,1,2,181 for 1 opponents:
  - Worse_and_Worse_2@_0 (score:2.015)

- C6,1,4,1,4,1,2,181 for 1 opponents:
  - Worse_and_Worse_2@_1 (score:2.135)

- C5,1,35,1,1,1,9,1,3,1,1,1,2,2,1,4,1,2,2,1,2,1,1,2,1,3,5,3,2,1,4,1,2,1,1,2,1,2,1,1,2,4,2,1,1,1,4,1,1,2,4,1,4,1,2,1,3,2,9,2,1,1,3,1,1,1,1,1,1 for 1 opponents:
  - Stalker@_9@_(D,) (score:3.66)

- C5,1,5,1,55,1,20,2,4,2,1,5,2,1,2,1,2,2,1,1,2,6,1,2,3,1,2,2,1,1,4,4,7,1,3,1,2,4,3,1,1,2,1,1,1,1,2,1,2,1,2,3,1,1,3,1,1,2,3,5 for 1 opponents:

  - Evolved_ANN_5_Noise_05 (score:3.58)

- C5,1,3,2,1,1,1,1,3,1,1,1,5,1,1,1,1,1,3,3,2,1,1,3,4,1,2,1,3,2,1,4,1,5,7,3,5,1,2,1,2,2,6,2,1,2,2,2,2,1,2,1,3,2,1,2,4,1,1,1,1,1,1,1,5,2,1,1,1,1 for 1 opponents:
  - Yamachi (score:3.9)

- C5,1,1,1,2,1,4,1,4,2,5,1,1,1,3,1,2,2,1,3,1,1,1,2,1,1,2,1,2,1,2,1,1,2,2,1,1,1,1,1,3,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,3 for 1 opponents:
  - Eatherley@_8 (score:3.535)

- C5,4,2,3,4,1,2,3,1,1,3,1,1,2,86,1,11,1,61,7 for 1 opponents:
  - ZD-Extortion@_3 (score:1.415)

- C5,11,2,8,3,1,1,69,19,1,28,1,50,1 for 1 opponents:
  - Stochastic_Cooperator@_3 (score:2.48)

- C5,195 for 3 opponents:
  - Appeaser (score:3.01)
  - GrudgerAlternator (score:3.01)
  - Win-Stay_Lose-Shift@_C (score:3.01)

- C4,1,72,1,2,1,33,1,14,1,31,1,34,1,2,1 for 1 opponents:
  - ZD-Mem2@_6 (score:2.685)

- C4,1,56,1,26,1,1,1,9,55,1,44 for 1 opponents:
  - Average_Copier@_0 (score:3.51)

- C4,1,39,1,55,100 for 1 opponents:
  - Dynamic_Two_Tits_For_Tat@_7 (score:3.44)

- C4,1,5,1,188,1 for 1 opponents:
  - Remorseful_Prober@_5 (score:2.6)

- C4,1,5,1,5,1,5,1,5,1,3,1,5,1,5,1,4,1,1,1,5,1,4,1,5,1,5,1,5,1,5,1,1,1,4,1,5,1,5,1,5,1,5,1,5,1,5,1,5,1,5,1,3,1,5,1,5,1,5,1,5,1,5,1,5,1,5,1 for 1 opponents:
  - DoubleResurrection (score:2.835)

- C4,1,4,2,2,2,2,1,1,1,1,1,7,1,3,1,1,1,2,1,1,4,1,2,2,2,1,2,7,2,2,1,1,1,2,1,1,1,124,3 for 1 opponents:
  - ZD-GEN-2@_1 (score:3.14)

- C4,1,2,1,1,3,4,2,1,1,2,1,1,1,1,2,1,1,1,1,2,1,2,1,3,1,4,2,1,1,1,2,1,1,2,1,1,1,1,1,2,1,1,4,1,3,3,3,1,2,1,1,1,2,3,2,1,1,1,2,1,1,1,2,1,6,1,1,2,2 for 1 opponents:

- SelfSteem@_9 (score:2.725)

- C4,89,1,106 for 1 opponents:
  - Stochastic_WSLS@_7 (score:3.01)

- C3,1,44,1,4,1,15,1,12,2,10,1,16,1,12,1,71,4 for 1 opponents:
  - ZD-Extort-2@_5 (score:2.2)

- C3,1,10,1,37,1,32,1,33,1,11,1,16,1,49,2 for 1 opponents:
  - Forgetful_Fool_Me_Once@_3 (score:3.09)

- C3,1,6,1,38,1,2,2,1,1,1,1,4,1,2,3,1,3,1,2,2,1,1,1,1,1,3,1,1,1,1,3,2,1,2,1,3,1,2,4,1,1,2,1,1,1,4,2,5,1,7,2,1,1,7,2,2,1,4,1,1,1,3,1,2,1,1,1,1
  for 1 opponents:
  - Stalker@_8@_(D,) (score:3.655)

- C3,1,2,1,1,1,2,1,63,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
  for 1 opponents:
  - Eatherley@_3 (score:3.4)

- C3,1,2,2,1,2,3,3,2,5,3,2,169,2 for 1 opponents:
  - Cave@_7 (score:3.105)

- C3,1,2,2,1,4,4,1,1,2,29,1,148,1 for 1 opponents:
  - Cave@_6 (score:3.035)

- C3,1,1,1,3,1,3,1,1,1,2,1,3,1,1,1,2,1,3,1,1,1,2,1,3,1,1,1,1,1,1,1,2,1,3,1,1,1,1,1,1,1,3,1,3,1,2,1,4,1,4,1,3,1,3,1,3,1,1,1,2,1,3,1,1,1,1,1,2,1
  for 1 opponents:
  - Evolved_FSM_16_Noise_05 (score:3.66)

- C3,2,45,1,29,1,24,1,6,1,1,1,14,1,63,7 for 1 opponents:
  - ZD-Extort-2_v2@_6 (score:2.16)

- C3,2,26,1,6,1,30,1,13,1,10,1,102,3 for 1 opponents:
  - ZD-Extort-2_v2@_5 (score:2.24)

- C3,2,4,2,1,2,2,2,3,1,4,2,2,1,1,1,3,3,2,1,1,1,3,1,1,1,1,1,4,2,2,2,1,2,2,1,2,1,1,1,3,2,3,2,2,1,1,2,4,2,3,2,1,1,2,1,2,1,3,1,2,1,2,5,2,1,2,1,4,1
  for 1 opponents:
  - Hard_Go_By_Majority@_20 (score:3.815)

- C3,2,3,3,1,1,1,2,1,2,3,3,5,4,1,5,2,1,1,1,1,1,2,1,1,1,7,1,1,1,2,1,3,1,2,2,5,1,1,5,3,1,1,3,2,1,2,1,3,4,1,1,3,4,1,2,2,2,1,4,3,2,4,1,1,1,1,1,4,3
  for 1 opponents:

  - ZD-Extort3@_5 (score:1.925)

- C3,2,2,1,3,1,3,1,1,2,3,1,3,1,3,3,4,1,2,3,6,3,3,1,3,1,1,1,2,2,3,1,1,1,2,1,4,1,1,2,1,1,5,2,4,1,2,2,1,1,4,1,3,1,1,1,2,1,2,1,3,1,2,2,2,1,3,1,3,2 for 1 opponents:
  - Hard_Go_By_Majority@_10 (score:3.695)

- C3,2,1,4,2,2,4,3,4,3,3,4,1,2,2,1,5,1,3,1,3,5,1,2,1,3,2,2,1,2,3,5,1,1,1,1,2,3,2,1,1,1,2,1,1,3,1,4,2,1,1,1,1,5,1,2,2,5,2,2,3,1,1,2,2,1,3,2,1,1 for 1 opponents:
  - ZD-Mischief@_5 (score:1.215)

- C3,6,24,1,25,1,8,1,11,1,9,1,14,1,23,1,4,1,21,1,28,1,4,10 for 1 opponents:
  - Evolved_ANN (score:3.26)

- C3,97,24,1,34,1,40 for 1 opponents:
  - Stochastic_Cooperator@_5 (score:2.345)

- C2,1,196,1 for 3 opponents:
  - Remorseful_Prober@_8 (score:2.655)
  - TF1 (score:3.0)
  - Tester (score:3.005)

- C2,1,97,26,1,73 for 1 opponents:
  - Average_Copier@_2 (score:3.495)

- C2,1,61,1,132,3 for 1 opponents:
  - Cave@_2 (score:3.02)

- C2,1,53,1,22,1,9,1,27,1,8,1,14,1,49,1,6,2 for 1 opponents:
  - Forgetful_Fool_Me_Once@_4 (score:3.1)

- C2,1,4,1,125,1,63,3 for 1 opponents:
  - Getzler@_1 (score:3.045)

- C2,1,4,1,18,1,73,1,97,2 for 1 opponents:
  - Forgetful_Fool_Me_Once@_7 (score:3.06)

- C2,1,4,1,4,2,2,184 for 1 opponents:
  - Worse_and_Worse_2@_9 (score:2.235)

- C2,1,2,1,2,1,1,3,2,1,1,2,2,1,2,2,1,1,1,1,1,2,3,1,1,1,1,1,3,2,3,3,3,2,2,1,1,2,2,2,3,2,2,1,2,1,4,2,1,1,1,3,3,2,1,1,3,1,1,4,1,1,3,1,4,1,1,1,1,1 for 1 opponents:

- Hard_Go_By_Majority@_40 (score:3.895)

- C2,1,2,1,1,1,2,1,1,1,3,2,5,2,4,1,2,1,1,1,2,1,1,1,2,1,1,1,2,1,1,1,3,1,3,1,3,1,1,1,2,1,1,1,2,1,1,1,2,1,3,2,3,1,2,1,1,1,2,1,1,1,2,1,1,1,3,2,5,2
  for 1 opponents:
  - Hard_Go_By_Majority@_5 (score:3.655)
- C2,1,1,1,194,1 for 1 opponents:
  - Calculator@_7 (score:3.0)
- C2,1,1,1,192,3 for 1 opponents:
  - ZD-GTFT-2@_2 (score:3.02)
- C2,1,1,1,72,1,121,1 for 2 opponents:
  - Joss@_6 (score:2.665)
  - Naive_Prober@_6 (score:2.665)
- C2,1,1,3,192,1 for 1 opponents:
  - Omega_TFT@_3,_8 (score:3.015)
- C2,1,1,196 for 1 opponents:
  - EvolvedLookerUp2_2_2 (score:4.955)
- C2,2,80,5,11,100 for 1 opponents:
  - Dynamic_Two_Tits_For_Tat@_4 (score:3.51)
- C2,2,45,1,5,1,48,1,50,1,35,9 for 1 opponents:
  - ZD-Extort3@_0 (score:1.93)
- C2,2,18,1,26,1,5,1,48,1,50,1,41,3 for 1 opponents:
  - ZD-Extort-2@_0 (score:2.21)
- C2,2,18,1,26,1,5,1,48,1,50,1,21,1,13,9 for 1 opponents:
  - ZD-Extort-2_v2@_0 (score:2.16)
- C2,2,2,2,1,3,2,2,2,2,2,2,2,2,2,2,2,2,14,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,3,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,3,2,2,2,2,2,2
  for 1 opponents:
  - Leyvraz@_8 (score:3.22)
- C2,3,82,3,2,3,2,1,2,3,2,2,3,2,2,1,4,2,1,1,1,1,2,1,4,6,1,1,1,1,1,5,7,1,1,2,11,2,2,4,2,2,4,1,1,4,2,1,5 for 1 opponents:

  - ZD-Extortion@_9 (score:1.74)

- C2,3,54,1,36,1,29,1,23,1,31,1,15,2 for 1 opponents:
  - Forgetful_Fool_Me_Once@_9 (score:3.1)

- C2,3,22,1,27,2,1,1,1,2,2,2,2,1,2,3,3,2,1,2,1,1,2,1,2,1,1,1,2,5,3,2,1,1,1,1,3,5,1,6,1,4,1,1,2,2,1,1,1,1,1,1,1,2,2,2,2,1,1,1,1,1,1,3,1,1,2, for 1 opponents:
  - ZD-Extort-2@_6 (score:2.325)

- C2,3,22,1,22,1,4,2,1,2,3,1,2,6,1,1,3,2,1,1,3,1,2,1,1,1,3,1,3,3,1,1,1,3,3,4,1,1,1,1,2,1,1,2,2,1,1,1,1,4,2,1,1,3,1,1,2,2,1,1,1,3,1,4,4,3,1,1, for 1 opponents:
  - ZD-Extort3@_6 (score:2.07)

- C2,3,3,2,2,1,1,2,1,1,3,2,175,2 for 1 opponents:
  - ZD-GTFT-2@_7 (score:3.03)

- C2,3,1,2,2,1,5,2,1,2,3,1,2,1,3,1,3,1,1,1,2,1,1,1,2,6,1,1,2,1,3,1,1,4,2,1,1,2,1,1,1,1,1,1,3,2,2,1,2,4,2,1,2,1,1,2,2,1,1,1,3,1,3,1,2,2,1,3,3,4 for 1 opponents:
  - ZD-SET-2@_9 (score:2.745)

- C2,6,2,6,1,4,8,1,16,4,1,1,11,9,20,100,8 for 1 opponents:
  - Stochastic_Cooperator@_0 (score:2.66)

- C2,6,1,3,1,1,1,1,1,1,1,1,1,3,1,7,1,7,3,2,3,1,3,1,4,3,1,1,1,1,1,1,2,6,4,1,1,1,1,3,1,2,1,2,1,1,1,2,1,1,1,1,1,3,1,1,3,2,4,2,3,1,2,5,1,2,4,2,1,1 for 1 opponents:
  - ZD-SET-2@_5 (score:2.29)

- C2,13,1,21,1,47,1,1,4,3,1,1,4,3,3,1,4,3,2,4,2,6,1,1,1,3,2,1,1,1,2,2,1,1,3,1,2,1,1,2,3,2,6,3,1,6,3,4,1,1,1,1,1,3,1,1,1,2,1,3 for 1 opponents:
  - Bush_Mosteller@_3 (score:3.42)

- C2,198 for 2 opponents:
  - Adaptive (score:4.88)
  - Nydegger (score:4.98)

- C1,1,197,1 for 3 opponents:
  - CollectiveStrategy (score:3.0)
  - Gladstein (score:3.005)
  - Ripoff (score:3.005)

- C1,1,196,2 for 2 opponents:
  - PSO_Gambler_2_2_2@_5 (score:3.03)
  - PSO_Gambler_2_2_2@_9 (score:3.03)

- C1,1,5,1,3,2,5,1,2,1,1,4,1,5,2,3,2,1,1,3,3,2,2,1,1,3,1,1,1,1,1,2,2,2,2,2,2,2,2,3,1,1,1,4,2,1,1,2,6,1,2,2,1,1,2,3,1,1,4,1,1,3,1,1,1,1,2,1,1,1 for 1 opponents:

  - ZD-Extortion@_5 (score:1.415)

- C1,1,3,1,18,2,2,1,2,1,1,1,1,1,3,3,6,4,4,1,2,1,1,2,1,1,7,1,2,1,2,2,2,1,2,3,1,1,2,1,1,2,2,1,1,3,3,3,2,1,2,1,1,1,1,1,1,1,1,2,1,1,3,1,4,1,6,1,1
  for 1 opponents:
  - Champion@_4 (score:3.82)

- C1,1,2,1,194,1 for 1 opponents:
  - Calculator@_1 (score:3.0)

- C1,1,2,1,162,1,30,2 for 1 opponents:
  - PSO_Gambler_2_2_2@_3 (score:3.05)

- C1,1,2,1,121,1,23,1,47,2 for 1 opponents:
  - Kluepfel@_9 (score:3.045)

- C1,1,2,1,9,1,22,1,11,2,1,1,2,1,4,2,6,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,11,3,2,2,3,2,2,1,9,5,1,4,1,3,1,1,1,2,1,1,3,1,2,2,6,1,2,1,1,1,1,2,2,1,3,1
  for 1 opponents:
  - Tranquilizer@_3 (score:3.25)

- C1,1,2,1,6,1,187,1 for 1 opponents:
  - Feld@_0 (score:2.335)

- C1,1,2,1,4,1,12,1,6,1,21,1,9,1,42,1,93,2 for 1 opponents:
  - PSO_Gambler_2_2_2@_0 (score:3.1)

- C1,1,2,1,4,1,7,1,1,1,2,3,2,1,53,1,4,1,2,2,1,2,1,1,1,2,5,2,5,3,2,1,4,1,2,3,6,1,1,1,3,2,4,7,6,1,1,1,5,2,1,1,2,2,3,1,1,1,3,1,2,1,2,1,1,1,1,3,2
  for 1 opponents:
  - Tranquilizer@_8 (score:3.32)

- C1,1,2,1,2,1,17,1,16,1,50,1,104,2 for 1 opponents:
  - PSO_Gambler_2_2_2@_7 (score:3.08)

- C1,1,2,1,1,1,3,1,1,1,2,1,1,1,2,1,3,2,3,1,2,1,1,1,3,2,4,2,3,1,1,1,3,1,3,2,3,2,3,1,3,2,3,1,1,1,2,1,2,1,1,1,2,1,2,1,1,1,3,1,2,1,2,1,1,1,3,1,1,1
  for 1 opponents:
  - Soft_Go_By_Majority@_5 (score:3.69)

- C1,1,2,1,1,1,3,2,3,6,1,2,1,1,2,1,3,1,3,3,2,2,4,2,3,5,2,2,1,2,3,2,2,2,2,2,1,1,12,1,34,1,42,1,26,4 for 1 opponents:
  - ZD-GEN-2@_9 (score:3.15)

- C1,1,2,1,1,2,1,1,2,1,1,1,1,2,1,1,2,1,1,4,6,4,4,1,1,1,1,1,1,1,1,2,2,1,1,1,4,1,1,2,1,1,2,3,2,1,1,1,2,1,1,2,1,1,1,1,2,1,1,1,3,2,2,3,4,1,7,1,1,3,1
  for 1 opponents:

- SelfSteem@_8 (score:2.76)

- C1,1,1,1,195,1 for 1 opponents:
  - Kluepfel@_7 (score:3.015)
- C1,1,1,1,115,1,79,1 for 1 opponents:
  - Remorseful_Prober@_2 (score:2.665)
- C1,1,1,1,28,1,1,1,87,1,1,1,57,1,1,1,14,1 for 1 opponents:
  - Weiner (score:3.03)
- C1,1,1,1,10,1,152,1,30,2 for 1 opponents:
  - Getzler@_3 (score:3.045)
- C1,1,1,1,2,1,1,4,1,1,1,1,2,1,1,1,1,1,1,2,1,1,4,1,1,3,3,1,1,3,2,5,1,2,1,2,1,2,3,1,1,2,3,2,1,1,3,3,1,1,1,2,1,1,3,1,1,1,3,1,1,2,2,1,1,4,1,3,1,3 for 1 opponents:
  - ZD-SET-2@_7 (score:2.555)
- C1,1,1,1,1,1,1,1,2,1,115,1,68,5 for 1 opponents:
  - Feld@_9 (score:2.31)
- C1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1 for 17 opponents:
  - Adaptive_Pavlov_2011 (score:3.97)
  - GTFT@_0.33 (score:3.085)
  - Harrington@_5 (score:3.205)
  - Harrington@_7 (score:3.205)
  - Random_Tit_for_Tat@_0 (score:3.055)
  - Random_Tit_for_Tat@_1 (score:3.205)
  - Random_Tit_for_Tat@_2 (score:3.13)
  - Random_Tit_for_Tat@_3 (score:3.16)
  - Random_Tit_for_Tat@_4 (score:3.31)
  - Random_Tit_for_Tat@_5 (score:3.25)
  - Random_Tit_for_Tat@_6 (score:3.235)
  - Random_Tit_for_Tat@_7 (score:3.31)
  - Random_Tit_for_Tat@_8 (score:3.37)
  - Random_Tit_for_Tat@_9 (score:3.355)
  - Soft_Go_By_Majority@_10 (score:4.0)
  - Soft_Go_By_Majority@_20 (score:4.0)
  - Soft_Go_By_Majority@_40 (score:4.0)
- C1,1,1,2,54,1,2,1,17,1,31,1,49,1,33,1,2,1 for 1 opponents:
  - Forgetful_Fool_Me_Once@_6 (score:3.1)
- C1,1,1,2,26,1,16,1,15,7,3,1,4,1,2,8,1,89,1,9,10 for 1 opponents:
  - Stochastic_Cooperator@_4 (score:2.815)
- C1,1,1,4,1,18,7,2,31,21,4,103,6 for 1 opponents:

  - Stochastic_Cooperator@_7 (score:2.855)

- C1,1,1,197 for 1 opponents:
  - Evolved_FSM_16 (score:3.665)
- C1,2,196,1 for 1 opponents:
  - TF2 (score:2.99)
- C1,2,2,1,2,1,1,2,58,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,11,87,1,3,1,8 for 1 opponents:
  - Eatherley@_4 (score:3.535)
- C1,2,2,1,2,1,1,2,1,3,3,3,161,1,14,2 for 1 opponents:
  - ZD-GEN-2@_4 (score:3.095)
- C1,2,2,81,1,30,1,82 for 1 opponents:
  - Stochastic_WSLS@_4 (score:3.05)
- C1,2,1,1,86,1,34,1,2,1,67,3 for 1 opponents:
  - ZD-Extort-2_v2@_9 (score:2.475)
- C1,2,1,1,4,1,1,2,1,1,1,2,1,1,1,3,2,2,1,1,1,4,1,2,1,1,1,2,1,4,1,1,1,2,1,1,2,3,1,3,2,2,1,1,2,1,3,2,1,1,1,2,1,1,2,3,1,6,2,1,2,2,2,2,4,1,1,1,1,3 for 1 opponents:
  - SelfSteem@_6 (score:2.65)
- C1,2,1,1,1,1,1,3,1,4,2,3,1,1,1,4,6,3,4,4,3,2,3,3,4,3,8,1,11,3,1,7,4,1,1,1,1,4,1,1,2,2,2,4,1,1,1,2,2,3,2,1,4,1,1,3,2,2,1,1,1,1,1,1,2,3,7,1,1 for 1 opponents:
  - ZD-Extortion@_4 (score:1.68)
- C1,2,1,1,1,2,3,2,3,1,2,4,1,1,1,2,3,1,4,2,1,1,1,1,2,1,1,2,3,2,3,2,1,1,1,2,1,1,1,2,1,4,1,1,1,1,2,1,1,1,4,2,1,1,2,1,1,4,1,1,1,2,3,1,2,1,1,2,1,1 for 1 opponents:
  - SelfSteem@_1 (score:2.81)
- C1,2,1,1,1,2,3,2,1,1,1,2,3,2,1,1,1,2,1,1,1,2,1,1,2,1,1,2,1,1,1,1,2,1,1,1,2,1,1,1,2,1,1,1,2,1,1,1,2,1,1,1,1,1,2,1,1,1,1,2,1,1,1,1,2,1,1,2,4,1,3,2,1,4,1,3,1,1,2,2 for 1 opponents:
  - SelfSteem@_5 (score:2.64)
- C1,2,1,1,1,2,3,2,1,1,1,2,1,1,1,2,2,3,1,1,1,2,1,1,2,3,1,2,1,1,1,1,4,1,2,1,4,1,1,5,1,1,3,1,1,2,2,2,2,2,1,1,2,1,1,1,4,1,1,1,1,3,3,6,1,1,4,2,1,1 for 1 opponents:
  - SelfSteem@_2 (score:2.615)
- C1,2,1,1,1,2,1,4,1,1,2,1,1,1,2,1,1,1,1,1,2,1,1,1,2,1,1,2,1,1,2,1,1,1,2,3,1,4,2,1,1,1,2,4,1,4,1,1,1,1,1,2,4,1,1,1,2,4,3,1,2,1,1,1,2,1,2,3,1,1,3,1,4 for 1 opponents:

  - SelfSteem@_3 (score:2.66)

- C1,2,1,1,1,4,1,2,1,2,1,1,1,2,1,1,1,1,2,1,1,1,1,2,1,1,1,4,1,1,2,1,1,2,1,1,2,1,1,1,2,3,2,1,4,1,2,1,1,1,1,1,1,2,1,1,1,2,1,4,1,1,2,1,2,1,1,1,1,3,1,2,2 for 1 opponents:
  - SelfSteem@_0 (score:2.615)

- C1,2,1,4,1,1,4,1,1,4,1,3,3,4,1,4,1,2,1,3,1,7,1,1,4,2,3,1,1,2,1,1,2,1,1,2,1,1,5,2,1,2,1,2,2,4,2,1,4,1,1,3,4,1,1,1,1,1,3,3,1,3,3,3,11,3,2,3,1,1,3,4 for 1 opponents:
  - ZD-Mischief@_6 (score:1.28)

- C1,2,1,4,1,1,2,1,1,1,1,4,2,2,2,1,3,1,1,1,165,2 for 1 opponents:
  - ZD-GEN-2@_8 (score:3.1)

- C1,2,1,4,1,1,1,3,2,4,1,4,1,2,4,1,1,1,2,1,1,1,1,2,1,1,1,2,1,1,1,1,1,2,1,2,1,2,3,1,1,2,1,1,1,2,1,1,1,1,2,1,1,1,2,2,1,1,3,1,2,3,4,1,4,3,2,1,4,2 for 1 opponents:
  - SelfSteem@_7 (score:2.715)

- C1,2,1,62,1,2,1,2,1,2,4,1,1,3,2,1,1,2,2,4,1,1,3,2,2,1,1,1,2,1,1,3,2,1,1,1,1,1,1,4,1,1,1,2,1,1,2,1,2,1,2,1,1,3,2,1,3,1,1,9,2,9,1,2,2,1,2,1,1,3,1 for 1 opponents:
  - Bush_Mosteller@_4 (score:3.5)

- C1,2,1,196 for 1 opponents:
  - Stochastic_WSLS@_8 (score:3.02)

- C1,3,2,2,18,1,168,5 for 1 opponents:
  - Getzler@_7 (score:3.095)

- C1,3,1,1,42,1,1,1,5,1,43,100 for 1 opponents:
  - Champion@_0 (score:3.87)

- C1,3,1,3,1,2,1,5,4,2,5,2,167,3 for 1 opponents:
  - Leyvraz@_7 (score:3.105)

- C1,3,1,3,1,5,1,5,1,7,1,21,1,6,1,3,3,1,1,8,1,1,1,9,2,1,1,1,1,5,2,4,1,2,1,4,1,1,1,3,2,1,1,1,1,6,4,3,1,1,3,1,1,1,1,1,1,2,3,1,1,4,1,2,1,3,3,4,2 for 4 opponents:
  - Arrogant_QLearner@_7 (score:4.225)
  - Cautious_QLearner@_7 (score:4.225)
  - Hesitant_QLearner@_7 (score:4.225)
  - Risky_QLearner@_7 (score:4.225)

- C1,4,3,1,3,1,46,1,33,1,103,3 for 1 opponents:

  - ZD-Extortion@_7 (score:1.695)

- C1,7,1,1,2,1,1,1,1,2,1,1,2,3,2,1,1,1,2,1,2,1,3,3,1,1,1,2,2,3,2,2,1,4,3,1,1,1,1,1,2,2,1,1,2,1,1,1,5,1,2,1,1,1,1,1,102,5 for 1 opponents:
  - ZD-Extortion@_8 (score:1.63)

- C1,45,1,35,1,6,2,1,1,1,1,5,1,4,1,1,2,7,5,1,2,4,2,2,1,4,1,1,1,5,2,5,2,1,4,1,4,2,1,6,1,2,1,1,2,1,1,2,1,2,5,5,3 for 1 opponents:
  - Bush_Mosteller@_8 (score:3.52)

- C1,140,1,58 for 1 opponents:
  - Stochastic_WSLS@_1 (score:3.06)

- C1,199 for 11 opponents:
  - Defector_Hunter (score:4.99)
  - Evolved_FSM_4 (score:3.67)
  - Handshake (score:4.97)
  - Opposite_Grudger (score:4.975)
  - SolutionB1 (score:4.955)
  - Tricky_Defector (score:4.915)
  - Willing@_0 (score:4.975)
  - Willing@_2 (score:4.975)
  - Willing@_5 (score:4.975)
  - Willing@_6 (score:4.975)
  - Win-Shift_Lose-Stay@_D (score:4.975)

- D1,198,1 for 21 opponents:
  - Calculator@_2 (score:2.945)
  - Calculator@_3 (score:2.99)
  - Calculator@_4 (score:2.975)
  - Calculator@_5 (score:2.945)
  - Calculator@_6 (score:3.005)
  - Calculator@_8 (score:2.975)
  - Calculator@_9 (score:3.005)
  - Evolved_HMM_5@_0 (score:3.02)
  - Evolved_HMM_5@_1 (score:3.02)
  - Evolved_HMM_5@_2 (score:3.02)
  - Evolved_HMM_5@_3 (score:3.02)
  - Evolved_HMM_5@_4 (score:3.02)
  - Evolved_HMM_5@_5 (score:3.02)
  - Evolved_HMM_5@_6 (score:3.02)
  - Evolved_HMM_5@_7 (score:3.02)
  - Evolved_HMM_5@_8 (score:3.02)
  - Evolved_HMM_5@_9 (score:3.02)
  - PSO_Gambler_1_1_1@_1 (score:3.02)
  - PSO_Gambler_1_1_1@_8 (score:3.02)
  - Winner12 (score:3.02)
  - Winner21 (score:3.0)

- D1,196,3 for 1 opponents:
  - PSO_Gambler_1_1_1@_3 (score:3.04)

- D1,194,5 for 2 opponents:
  - Feld@_7 (score:2.38)
  - ZD-GTFT-2@_4 (score:3.025)

- D1,193,1,4,1 for 1 opponents:
  - ZD-Extort-4@_8 (score:1.785)

- D1,99,1,88,1,8,2 for 1 opponents:

  - Remorseful_Prober@_7 (score:2.78)

- D1,99,100 for 1 opponents:
  - Revised_Downing@_True (score:4.01)

- D1,79,1,117,2 for 1 opponents:
  - Remorseful_Prober@_6 (score:2.67)

- D1,76,1,34,1,86,1 for 1 opponents:
  - ZD-Extort-4@_6 (score:1.69)

- D1,70,1,127,1 for 1 opponents:
  - Feld@_1 (score:2.37)

- D1,68,1,10,1,13,1,101,4 for 1 opponents:
  - Getzler@_5 (score:3.03)

- D1,60,1,95,1,39,3 for 1 opponents:
  - Forgetful_Fool_Me_Once@_0 (score:3.06)

- D1,45,1,35,1,19,1,34,1,60,2 for 1 opponents:
  - Forgetful_Fool_Me_Once@_8 (score:3.07)

- D1,42,1,7,1,5,2,3,1,1,1,2,1,2,1,1,1,1,1,1,4,1,1,1,1,2,3,2,1,1,5,2,2,2,3,1,1,2,2,2,1,1,2,1,7,2,2,1,1,3,2,1,3,3,1,8,1,1,2,1,1,1,1,2,3,3,1,1,3 for 1 opponents:
  - Champion@_2 (score:3.81)

- D1,21,1,20,1,75,1,6,1,24,1,46,2 for 1 opponents:
  - Forgetful_Fool_Me_Once@_2 (score:3.08)

- D1,14,1,183,1 for 1 opponents:
  - ZD-Extort-4@_1 (score:1.785)

- D1,10,1,25,1,14,1,78,1,16,1,49,2 for 1 opponents:
  - ZD-Mem2@_3 (score:2.605)

- D1,9,1,1,1,1,1,3,1,3,1,7,1,19,1,6,1,13,1,11,1,24,1,24,1,7,1,11,1,44,2 for 1 opponents:
  - Forgetful_Fool_Me_Once@_1 (score:3.17)

- D1,7,1,1,1,7,182 for 1 opponents:

  - Worse_and_Worse_2@_2 (score:2.085)

- D1,7,11,1,1,1,3,1,2,1,28,1,75,1,7,1,11,1,46 for 1 opponents:
  - Stochastic_Cooperator@_1 (score:2.555)

- D1,7,192 for 1 opponents:
  - Grofman@_2 (score:3.31)

- D1,5,1,1,1,1,2,4,2,3,1,1,3,45,1,11,1,113,3 for 1 opponents:
  - Cave@_1 (score:3.07)

- D1,5,2,6,1,1,1,5,2,5,1,2,1,4,1,3,1,2,1,4,1,2,1,7,1,1,1,6,1,2,1,3,1,6,2,6,1,2,1,3,1,2,1,4,1,3,1,5,1,2,1,6,1,1,1,6,2,6,1,2,1,6,1,1,1,7,2,6,1,1 for 1 opponents:
  - MoreGrofman (score:3.49)

- D1,5,2,1,2,1,1,2,3,1,1,2,1,2,2,2,3,3,5,7,8,1,5,1,2,1,2,2,1,1,1,1,1,1,1,1,1,1,5,1,1,2,1,2,3,1,2,4,1,5,1,1,3,1,1,2,5,3,1,1,2,4,1,2,1,1,1,2,2,1 for 1 opponents:
  - ZD-SET-2@_3 (score:2.385)

- D1,5,194 for 2 opponents:
  - Grofman@_3 (score:3.27)
  - Grofman@_9 (score:3.41)

- D1,4,1,90,1,29,1,23,1,18,1,24,1,4,1 for 1 opponents:
  - ZD-Mem2@_9 (score:2.79)

- D1,4,1,4,1,21,1,20,1,26,1,13,1,29,1,34,1,34,6 for 1 opponents:
  - Forgetful_Fool_Me_Once@_5 (score:3.11)

- D1,4,1,1,2,4,1,14,2,10,1,1,1,14,1,5,1,11,1,13,1,4,1,14,1,13,1,23,1,15,1,20,1,2,1,1,11 for 1 opponents:
  - Evolved_ANN_5 (score:3.31)

- D1,4,2,5,1,2,1,2,2,3,1,1,4,65,1,1,2,5,3,2,2,1,2,3,1,7,2,5,3,3,3,1,3,4,1,4,1,1,2,3,1,3,2,1,1,3,2,4,1,1,3,2,3,1,1,3,4,4 for 1 opponents:
  - Tranquilizer@_2 (score:3.325)

- D1,4,20,1,174 for 2 opponents:
  - Grofman@_6 (score:3.31)
  - Grofman@_7 (score:3.37)

- D1,4,178,1,16 for 1 opponents:

- Grofman@_4 (score:3.43)

- D1,4,195 for 4 opponents:
  - Grofman@_0 (score:3.22)
  - Grofman@_1 (score:3.3)
  - Grofman@_5 (score:3.38)
  - Grofman@_8 (score:3.44)

- D1,3,1,172,1,21,1 for 1 opponents:
  - ZD-Extort-4@_0 (score:1.705)

- D1,3,1,80,1,1,1,3,1,5,2,2,1,1,1,1,2,1,1,1,2,2,2,1,4,2,1,1,2,3,3,1,1,1,3,1,1,2,3,1,1,2,1,1,4,2,1,2,1,1,1,1,1,1,1,1,1,2,2,1,1,2,1,3,2,1,1,1
for 1 opponents:
  - Feld@_4 (score:2.225)

- D1,3,1,52,1,50,1,24,1,7,1,11,1,43,1,1,1 for 1 opponents:
  - ZD-Mem2@_1 (score:2.775)

- D1,3,1,1,4,2,1,17,3,1,1,1,1,1,1,1,2,2,1,4,1,1,1,3,2,1,3,4,2,2,3,11,1,10,1,1,1,14,1,86,2 for 1 opponents:
  - ZD-Mem2@_5 (score:2.715)

- D1,3,196 for 4 opponents:
  - PSO_Gambler_2_2_2_Noise_05@_0 (score:3.655)
  - PSO_Gambler_2_2_2_Noise_05@_2 (score:3.655)
  - PSO_Gambler_2_2_2_Noise_05@_5 (score:3.655)
  - PSO_Gambler_2_2_2_Noise_05@_6 (score:3.655)

- D1,2,1,18,1,7,1,26,1,13,1,69,1,52,4,1,1 for 1 opponents:
  - ZD-Extort-2_v2@_1 (score:2.385)

- D1,2,1,2,90,1,1,1,2,2,1,1,1,1,1,1,1,1,3,1,3,1,2,3,3,2,1,1,2,2,2,2,2,2,2,2,1,1,4,1,1,2,3,1,2,1,3,2,2,1,2,2,5,1,2,1,2,1,2,1,1,3,1,1,3
for 1 opponents:
  - Bush_Mosteller@_9 (score:3.27)

- D1,2,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
for 1 opponents:
  - Black@_0 (score:3.39)

- D1,2,1,1,12,3,1,4,2,1,2,1,1,1,1,1,2,2,4,2,2,1,1,1,1,3,1,2,1,2,1,2,2,1,1,3,1,3,2,1,1,5,1,1,1,2,4,5,1,3,6,6,4,1,1,3,1,1,1,1,1,3,4,1,4,2,6,1,1
for 1 opponents:
  - ZD-Mischief@_7 (score:1.665)

- D1,2,2,59,1,61,1,24,1,43,5 for 1 opponents:

- ZD-Extortion@_2 (score:1.565)

- D1,2,2,17,1,7,1,26,1,13,1,69,1,52,4,1,1 for 1 opponents:

  - ZD-Extort-2@_1 (score:2.545)

- D1,2,2,9,1,22,1,47,1,33,1,47,1,30,2 for 1 opponents:

  - Kluepfel@_3 (score:3.04)

- D1,2,2,3,1,1,1,3,4,1,2,1,1,1,1,2,2,2,4,2,1,1,4,1,1,1,1,1,1,2,2,1,2,2,1,1,2,3,7,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4 for 1 opponents:

  - Grumpy@_Nice,_10,_-10 (score:4.05)

- D1,2,3,2,2,2,1,1,1,1,1,2,1,3,4,2,1,2,1,2,1,3,3,1,1,1,2,2,3,2,2,1,1,20,2,1,4,1,2,1,1,3,4,1,2,4,2,2,1,1,3,1,2,5,4,1,1,1,3,1,1,2,3,2,6,1,1,3,5 for 1 opponents:

  - ZD-Mischief@_8 (score:1.745)

- D1,2,197 for 9 opponents:

  - PSO_Gambler_2_2_2@_1 (score:3.025)
  - PSO_Gambler_2_2_2@_4 (score:3.065)
  - PSO_Gambler_2_2_2@_8 (score:3.025)
  - PSO_Gambler_2_2_2_Noise_05@_1 (score:3.665)
  - PSO_Gambler_2_2_2_Noise_05@_3 (score:3.665)
  - PSO_Gambler_2_2_2_Noise_05@_4 (score:3.665)
  - PSO_Gambler_2_2_2_Noise_05@_7 (score:3.665)
  - PSO_Gambler_2_2_2_Noise_05@_8 (score:3.665)
  - PSO_Gambler_2_2_2_Noise_05@_9 (score:3.665)

- D1,1,1,116,1,11,1,16,1,47,4 for 1 opponents:

  - ZD-Extort3@_3 (score:1.935)

- D1,1,1,82,1,33,1,28,1,47,4 for 1 opponents:

  - ZD-Extort-2_v2@_3 (score:2.235)

- D1,1,1,82,1,33,1,11,1,16,1,47,4 for 1 opponents:

  - ZD-Extort-2@_3 (score:2.32)

- D1,1,1,3,1,6,1,2,1,2,1,37,1,139,3 for 1 opponents:

  - Cave@_0 (score:3.04)

- D1,1,1,3,3,7,1,2,1,1,1,1,3,1,1,3,1,5,1,2,4,2,1,1,1,3,2,2,2,2,2,2,2,2,2,3,2,2,2,5,1,2,2,1,1,1,1,2,2,3,1,1,2,1,1,4,1,1,1,3,1,1,2,1,1,3,1,1,1,1 for 1 opponents:

  - Black@_1 (score:3.56)

- D1,1,1,2,2,1,1,2,1,3,5,21,2,24,2,22,1,4,1,29,1,2,1,20,1,31,1,11,1,2,3 for 1 opponents:

– ZD-Mischief@_9 (score:1.93)

- D1,1,1,1,1,194,1 for 1 opponents:
  – Calculator@_0 (score:2.98)
- D1,1,1,1,1,29,1,13,1,7,1,14,1,14,1,54,1,38,1,6,1,3,1,3,5 for 1 opponents:
  – ZD-Mem2@_4 (score:2.605)
- D1,1,1,1,1,22,1,25,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,29,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,1,1,1,1,1,1,1,1,1,3,1,1,1,1,1,1
  for 1 opponents:
  – RichardHufford (score:3.005)
- D1,1,1,1,1,20,1,173,1 for 1 opponents:
  – ZD-Extort-4@_7 (score:1.94)
- D1,1,1,1,1,17,1,41,1,54,1,31,1,46,2 for 1 opponents:
  – ZD-Extort-2_v2@_2 (score:2.155)
- D1,1,1,1,1,17,1,41,1,54,1,6,1,24,1,30,1,15,2 for 1 opponents:
  – ZD-Extort-2@_2 (score:2.205)
- D1,1,1,1,1,17,1,21,1,44,1,1,1,1,1,1,3,2,1,3,1,1,2,1,2,2,1,3,3,1,1,4,1,3,2,9,1,3,1,2,3,1,2,1,3,3,2,1,2,1,1,1,2,1,1,1,3,6,2,1,1,1,1,1,1,2,1,1
  for 1 opponents:
  – ZD-Extort3@_2 (score:2.125)
- D1,1,1,1,1,3,1,1,1,68,1,4,7,1,2,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
  for 1 opponents:
  – Tranquilizer@_7 (score:3.23)
- D1,1,1,1,1,2,1,1,1,1,2,10,1,2,1,9,1,2,1,60,1,88,1,9,1 for 1 opponents:
  – ZD-Mem2@_7 (score:2.83)
- D1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
  for 1 opponents:
  – Sneaky_Tit_For_Tat (score:3.265)
- D1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
  for 6 opponents:
  – Hard_Tit_For_2_Tats (score:4.01)
  – N_Tit(s)_For_M_Tat(s)@_3,_2 (score:4.01)
  – Once_Bitten (score:4.01)
  – Pun1 (score:3.99)
  – Slow_Tit_For_Two_Tats_2 (score:4.01)
  – Tit_For_2_Tats (score:4.01)

- D1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,4,2,1,5,2,5,2,2,1,3,2,1,2,1,2,3,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4
for 1 opponents:

  - Black@_6 (score:3.415)

- D1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
for 1 opponents:

  - Harrington@_6 (score:3.205)

- D1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
for 1 opponents:

  - Harrington@_2 (score:3.21)

- D1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1
for 1 opponents:

  - Harrington@_3 (score:3.21)

- D1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
for 1 opponents:

  - Harrington@_4 (score:3.205)

- D1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1
for 1 opponents:

  - Harrington@_9 (score:3.21)

- D1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1
for 1 opponents:

  - Harrington@_8 (score:3.21)

- D1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
for 1 opponents:

  - Harrington@_0 (score:3.205)

- D1,1,1,1,1,1,1,1,1,3,2,1,1,1,1,1,1,1,2,4,1,2,1,1,1,1,1,1,4,2,1,2,3,1,1,1,3,2,1,1,1,1,1,1,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
for 1 opponents:

  - Black@_9 (score:3.515)

- D1,1,1,1,2,1,1,1,2,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
for 1 opponents:

  - Harrington@_1 (score:3.21)

- D1,1,1,1,6,4,1,1,1,4,3,2,2,2,4,3,1,2,2,2,2,2,6,2,1,1,3,1,1,2,2,2,1,1,1,1,1,1,1,1,2,2,1,1,1,3,1,1,2,3,1,3,1,1,1,2,1,1,3,1,2,1,2,1,1,1,1,1,1,1
for 1 opponents:

  - ZD-Mischief@_2 (score:1.125)

- D1,1,2,143,1,49,3 for 1 opponents:
  - Getzler@_9 (score:3.055)
- D1,1,2,4,1,1,1,2,1,2,1,2,1,1,3,1,1,3,1,1,2,165,2 for 1 opponents:
  - Cave@_3 (score:3.06)
- D1,1,2,1,1,1,1,5,2,9,2,3,2,10,1,40,1,2,1,16,1,71,1,8,1,15,1 for 1 opponents:
  - ZD-Mem2@_8 (score:2.685)
- D1,1,3,2,2,1,1,3,4,1,2,2,1,4,3,1,1,2,4,1,2,2,1,1,4,1,2,4,1,1,1,3,2,7,1,6,1,6,1,4,1,2,1,2,1,1,1,1,7,1,4,2,5,1,3,1,1,2,1,1,6,1,1,1,1,1,1,1,3,1
  for 1 opponents:
  - ZD-Extortion@_6 (score:1.335)
- D1,1,3,1,1,2,2,1,1,3,2,1,1,1,1,4,1,2,1,1,1,1,1,1,1,2,1,1,1,2,1,1,1,1,1,1,1,1,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,1
  for 1 opponents:
  - Black@_3 (score:3.555)
- D1,1,4,1,4,1,2,1,2,1,1,1,1,1,9,1,2,1,1,1,10,1,8,1,9,1,3,2,2,1,4,1,3,1,5,1,7,2,1,1,2,1,6,1,2,1,2,1,3,1,4,1,4,2,3,1,2,2,4,1,1,1,2,1,3,5,1,1,3
  for 4 opponents:
  - Arrogant_QLearner@_9 (score:4.31)
  - Cautious_QLearner@_9 (score:4.31)
  - Hesitant_QLearner@_9 (score:4.31)
  - Risky_QLearner@_9 (score:4.31)
- D1,1,9,1,14,1,8,1,1,1,6,1,5,1,3,1,7,1,3,1,14,1,6,1,2,1,3,3,5,1,1,1,11,1,1,1,7,3,8,2,1,2,2,2,1,1,2,2,1,1,1,5,2,1,6,1,2,1,2,1,4,2,1,2,1,1,11
  for 4 opponents:
  - Arrogant_QLearner@_8 (score:4.255)
  - Cautious_QLearner@_8 (score:4.255)
  - Hesitant_QLearner@_8 (score:4.255)
  - Risky_QLearner@_8 (score:4.255)
- D2,197,1 for 3 opponents:
  - Fortress3 (score:2.99)
  - Prober (score:3.01)
  - Prober_3 (score:2.995)
- D2,196,2 for 2 opponents:
  - Kluepfel@_2 (score:3.04)
  - Leyvraz@_3 (score:3.025)
- D2,195,3 for 1 opponents:
  - ZD-GEN-2@_6 (score:3.015)
- D2,194,4 for 2 opponents:

  - Leyvraz@_4 (score:3.06)
  - ZD-GEN-2@_0 (score:3.025)

- D2,193,5 for 1 opponents:
  - PSO_Gambler_1_1_1@_4 (score:3.07)

- D2,4,1,3,2,1,1,2,1,2,1,2,1,2,2,1,1,1,1,4,1,4,1,1,6,1,1,1,2,5,4,6,1,3,1,1,1,1,2,1,1,3,1,1,2,1,1,2,1,1,3,1,2,1,2,1,2,2,2,1,3,1,2,1,4,1,3,3,1,2 for 1 opponents:
  - ZD-SET-2@_0 (score:2.315)

- D2,3,2,2,2,1,1,1,1,1,6,1,1,1,1,3,1,1,1,1,3,3,3,1,1,1,2,2,3,2,2,1,4,3,1,1,1,1,1,2,1,1,3,3,1,2,2,1,2,1,1,1,1,1,1,1,1,1,1,4,3,1,1,1,1,4,1,2,5,2,2 for 1 opponents:
  - ZD-Extort3@_8 (score:2.475)

- D2,3,2,2,4,1,4,3,2,5,1,1,1,1,1,2,1,3,3,1,1,1,2,2,3,2,2,1,2,3,2,1,1,1,4,1,1,2,1,3,1,2,2,1,2,1,1,1,1,1,1,1,1,1,1,4,3,1,1,1,1,4,1,2,5,2,2,1,5,1,4 for 1 opponents:
  - ZD-Extort-2@_8 (score:2.505)

- D2,3,3,2,1,2,1,1,1,1,1,1,1,2,3,2,1,13,1,155,3 for 1 opponents:
  - Cave@_8 (score:3.1)

- D2,3,46,1,125,1,22 for 1 opponents:
  - Stochastic_WSLS@_0 (score:3.09)

- D2,2,1,107,1,1,1,14,1,46,1,19,1,2,1 for 1 opponents:
  - Kluepfel@_6 (score:3.06)

- D2,2,1,17,1,26,1,7,1,97,1,43,1 for 1 opponents:
  - ZD-Mem2@_0 (score:2.62)

- D2,2,1,17,1,7,1,19,1,6,1,13,1,36,1,24,1,7,1,11,1,42,4 for 1 opponents:
  - ZD-Extort3@_1 (score:2.145)

- D2,2,1,4,1,1,2,3,1,1,1,1,3,1,3,1,1,5,2,1,7,5,3,3,1,1,7,2,1,5,1,3,2,2,2,1,1,5,1,1,1,1,5,1,1,1,2,1,5,3,1,2,1,2,3,2,1,1,6,8,4,1,8,3,6,1,3,2,1,1 for 1 opponents:
  - ZD-SET-2@_6 (score:2.375)

- D2,2,1,2,1,1,1,1,3,2,1,1,1,3,3,5,1,2,3,4,4,4,4,4,4,4,4,6,2,5,1,1,2,3,1,1,1,1,1,1,1,1,2,1,2,2,2,7,1,1,2,2,3,1,1,6,2,1,4,4,1,2,1,1,1,1,1,1,1,2 for 1 opponents:
  - Black@_5 (score:3.335)

- D2,2,2,33,1,42,1,2,1,12,1,3,1,34,1,36,1,19,1,1,1,1,2 for 1 opponents:

  - ZD-Extort-2_v2@_8 (score:2.37)

- D2,1,1,194,2 for 1 opponents:
  - Kluepfel@_8 (score:3.03)

- D2,1,1,58,1,99,1,13,1,20,3 for 1 opponents:
  - Getzler@_6 (score:3.055)

- D2,1,1,47,1,52,1,92,3 for 1 opponents:
  - Getzler@_0 (score:3.045)

- D2,1,1,2,1,1,3,1,1,1,2,1,2,1,1,3,3,1,1,1,2,1,1,4,1,3,1,41,3,2,2,2,3,1,1,3,1,3,3,1,1,2,1,1,4,2,3,1,3,1,1,2,2,1,4,2,5,4,1,3,3,2,5,2,1,1,1,2,1 for 1 opponents:
  - ZD-Extort3@_7 (score:2.27)

- D2,1,1,1,1,1,1,2,1,3,3,1,1,1,1,1,2,102,1,55,1,14,3 for 1 opponents:
  - Cave@_9 (score:3.07)

- D2,1,1,1,2,2,1,1,1,69,1,15,2,2,1,1,1,1,2,1,1,1,2,2,2,1,4,2,1,1,2,3,3,1,1,1,3,1,1,2,3,1,1,2,1,2,3,2,1,2,1,1,1,1,1,1,1,1,1,2,2,1,1,2,1,4,1, for 1 opponents:
  - ZD-Extort-4@_4 (score:1.805)

- D2,1,1,1,2,1,2,2,1,2,1,9,2,17,1,28,1,1,1,2,3,1,2,2,1,2,2,1,1,1,2,2,1,1,1,1,1,1,3,3,1,1,2,3,2,13,3,2,2,2,4,1,1,3,1,2,1,1,5,1,1,2,1,2,2,2,2,1 for 1 opponents:
  - ZD-Extort-2_v2@_7 (score:2.48)

- D2,1,1,1,6,1,73,1,1,2,7,1,3,1,1,1,2,2,4,2,4,2,4,2,2,1,2,1,2,2,6,1,2,1,2,1,3,2,1,1,1,1,2,5,1,4,2,2,6,3,1,3,3,1,1,3,2,4 for 1 opponents:
  - Bush_Mosteller@_0 (score:3.31)

- D2,1,2,193,2 for 2 opponents:
  - Leyvraz@_1 (score:3.03)
  - Leyvraz@_2 (score:3.025)

- D2,1,2,192,3 for 1 opponents:
  - Leyvraz@_6 (score:3.02)

- D2,1,2,186,9 for 1 opponents:
  - WmAdams@_7 (score:3.07)

- D2,1,2,4,1,187,3 for 1 opponents:

  - Getzler@_8 (score:3.03)

- D2,1,2,2,2,1,1,1,1,2,1,2,1,1,1,1,1,1,1,174,1 for 1 opponents:

  - PSO_Gambler_1_1_1@_7 (score:3.12)

- D2,1,2,2,5,1,5,1,9,1,2,1,8,1,1,1,8,1,2,1,5,1,5,1,5,1,9,1,6,2,5,1,9,1,1,1,5,1,10,1,5,1,4,1,1,1,9,1,5,1,1,1,5,1,8,1,7,1,4,1,1,1,6,1,1,1,1,1,2
for 1 opponents:

  - Adaptive_Pavlov_2006 (score:4.01)

- D2,1,2,1,1,1,24,1,20,1,57,1,88 for 1 opponents:

  - Stochastic_WSLS@_5 (score:3.02)

- D2,1,2,1,2,2,2,2,2,2,2,2,2,2,2,2,3,1,2,2,2,2,2,2,3,1,2,1,3,2,2,2,2,2,2,1,3,2,2,2,3,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,3,1,2,2,2,2,2,2
for 1 opponents:

  - Leyvraz@_5 (score:3.095)

- D2,1,2,1,2,1,2,4,1,2,2,4,2,1,1,1,1,1,2,1,1,2,1,3,1,3,1,4,1,2,4,1,1,1,1,3,2,2,2,2,2,2,2,2,2,2,2,2,2,2,4,2,2,2,4,1,2,1,1,2,2,1,1,1,3,4,4,4,2
for 1 opponents:

  - Black@_7 (score:3.59)

- D2,1,3,1,4,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
for 1 opponents:

  - Leyvraz@_9 (score:3.155)

- D2,1,4,4,1,3,1,1,1,2,2,3,5,1,1,1,1,2,6,1,4,2,1,1,1,1,1,1,2,3,1,1,1,2,2,1,4,1,1,1,1,1,4,3,2,1,2,2,2,1,3,4,3,1,1,1,2,1,1,2,1,1,1,1,2,2,4,1,5,1
for 1 opponents:

  - ZD-SET-2@_2 (score:2.5)

- D2,1,7,2,7,4,2,4,1,1,4,1,2,1,3,4,4,1,1,1,3,1,1,4,9,1,12,2,1,2,1,1,1,1,24,1,74,8 for 1 opponents:

  - ZD-SET-2@_4 (score:2.5)

- D2,1,19,1,96,1,31,1,46,1,1 for 1 opponents:

  - Stochastic_WSLS@_2 (score:3.07)

- D2,1,197 for 1 opponents:

  - Prober_2 (score:4.97)

- D3,196,1 for 4 opponents:

  - Fortress4 (score:2.98)
  - Hard_Prober (score:3.0)
  - Predator (score:3.0)
  - Raider (score:3.0)

- D3,194,3 for 1 opponents:

– Stein_and_Rapoport@_0.05@_(D,_D) (score:3.02)

- D3,192,5 for 8 opponents:
  - WmAdams@_0 (score:3.06)
  - WmAdams@_1 (score:3.06)
  - WmAdams@_2 (score:3.06)
  - WmAdams@_3 (score:3.06)
  - WmAdams@_5 (score:3.06)
  - WmAdams@_6 (score:3.06)
  - WmAdams@_8 (score:3.06)
  - WmAdams@_9 (score:3.06)
- D3,177,1,2,1,13,3 for 1 opponents:
  - Getzler@_4 (score:3.065)
- D3,97,100 for 1 opponents:
  - Champion@_1 (score:3.83)
- D3,56,1,36,1,32,1,6,1,45,1,9,2,1,5 for 1 opponents:
  - Tullock@_9 (score:2.875)
- D3,24,1,2,1,4,2,1,1,2,2,1,1,1,1,1,1,5,2,1,2,3,2,3,3,1,1,1,3,1,1,1,1,1,3,1,3,2,1,1,1,1,4,1,2,2,5,1,1,5,1,2,2,2,3,1,2,4,3,1,1,1,2,1,3,2,1,2 for 1 opponents:
  - Champion@_8 (score:3.8)
- D3,20,1,1,1,1,1,3,1,1,1,44,1,6,2,1,1,9,1,1,1,1,1,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, for 1 opponents:
  - Tranquilizer@_0 (score:3.245)
- D3,12,1,151,1,22,10 for 1 opponents:
  - ZD-Extort-4@_3 (score:1.79)
- D3,8,1,1,2,7,21,1,156 for 1 opponents:
  - Meta_Hunter_Aggressive@_7_players (score:2.615)
- D3,4,4,1,2,2,5,1,1,3,2,2,3,2,2,1,5,3,4,2,2,1,1,1,2,1,1,6,2,2,1,2,2,1,1,5,1,1,1,4,3,1,2,1,1,1,4,1,1,1,1,2,1,1,5,1,1,1,1,1,1,2,5,5,1,4,2,3,1,2 for 1 opponents:
  - ZD-SET-2@_1 (score:2.565)
- D3,2,9,2,3,2,1,3,2,1,1,7,4,4,1,2,1,2,3,1,1,1,2,1,2,1,2,1,2,2,1,1,4,2,1,1,1,1,2,2,1,1,1,1,1,2,1,2,1,5,1,4,2,1,1,1,1,3,2,2,3,2,1,1,1,2,4,1,1,3 for 1 opponents:
  - ZD-Mischief@_3 (score:1.23)
- D3,1,1,86,1,4,1,29,1,2,1,17,1,34,1,5,1,8,3 for 1 opponents:

  - ZD-Extort-2@_9 (score:2.555)

- D3,1,1,54,1,36,1,3,50,1,43,1,5 for 1 opponents:
  - Stochastic_Cooperator@_9 (score:2.75)

- D3,1,1,8,4,1,1,25,1,147,8 for 1 opponents:
  - Tullock@_7 (score:2.92)

- D3,1,1,4,2,2,2,2,1,1,1,2,2,3,2,1,1,1,1,2,2,2,7,28,1,2,2,4,1,1,1,4,1,1,2,3,1,1,1,1,3,1,1,2,1,2,4,6,2,4,2,2,1,4,1,3,1,2,5,3,1,2,1,2,3,1,3,3,2 for 1 opponents:
  - ZD-Mischief@_1 (score:1.655)

- D3,1,1,3,1,3,1,2,1,2,2,1,2,1,5,1,1,1,1,17,1,32,1,11,5,2,1,2,1,2,8,2,1,1,1,1,1,1,1,5,2,4,1,5,1,1,2,1,3,4,2,2,3,5,3,2,1,2,1,5,2,2,2,2,1,1,3,2 for 1 opponents:
  - ZD-Extortion@_1 (score:1.535)

- D3,1,1,1,1,2,1,1,1,2,1,1,2,1,1,1,2,1,1,1,2,1,1,1,1,2,1,1,2,1,1,1,2,1,1,2,1,1,2,1,1,2,1,1,1,4,1,1,2,4,1,1,1,2,1,1,1,2,1,2,1,1,1,2,1,1,3,4,1,2,1,4,1,1,2 for 1 opponents:
  - SelfSteem@_4 (score:2.64)

- D3,1,2,1,6,1,16,1,1,1,2,1,9,1,8,1,5,1,6,1,7,1,6,2,10,1,2,3,3,2,2,2,2,1,3,1,2,1,5,1,6,1,1,1,2,1,5,4,5,1,5,2,5,1,2,1,2,1,1,1,4,1,2,2,3,1,1,3, for 4 opponents:
  - Arrogant_QLearner@_6 (score:4.98)
  - Hopeless@_3 (score:5.0)
  - Hopeless@_4 (score:5.0)
  - Hopeless@_5 (score:4.98)
  - Hopeless@_6 (score:4.98)
  - Hopeless@_7 (score:5.0)
  - Hopeless@_8 (score:5.0)
  - Hopeless@_9 (score:5.0)
  - Knowledgeable_Worse_and_Worse@_0 (score:2.88)
  - Knowledgeable_Worse_and_Worse@_1 (score:3.04)
  - Knowledgeable_Worse_and_Worse@_2 (score:3.04)
  - Knowledgeable_Worse_and_Worse@_3 (score:2.82)
  - Knowledgeable_Worse_and_Worse@_4 (score:3.12)
  - Knowledgeable_Worse_and_Worse@_5 (score:2.74)
  - Knowledgeable_Worse_and_Worse@_6 (score:3.12)
  - Knowledgeable_Worse_and_Worse@_7 (score:3.14)
  - Knowledgeable_Worse_and_Worse@_8 (score:2.94)
  - Knowledgeable_Worse_and_Worse@_9 (score:3.18)
  - Math_Constant_Hunter (score:5.0)
  - Negation@_0 (score:4.98)
  - Negation@_1 (score:5.0)
  - Negation@_2 (score:4.98)
  - Negation@_3 (score:5.0)
  - Negation@_4 (score:5.0)
  - Negation@_5 (score:4.98)
  - Negation@_6 (score:4.98)
  - Negation@_7 (score:5.0)
  - Negation@_8 (score:5.0)
  - Negation@_9 (score:5.0)
  - Random@_0 (score:2.86)
  - Random@_1 (score:3.02)
  - Random@_2 (score:2.96)
  - Random@_3 (score:2.92)
  - Random@_4 (score:3.04)
  - Random@_5 (score:2.82)
  - Random@_6 (score:3.0)
  - Random@_7 (score:3.2)
  - Random@_8 (score:3.14)
  - Random@_9 (score:3.1)
  - Random_Hunter (score:5.0)
  - ThueMorse (score:3.0)
  - ThueMorseInverse (score:3.0)
  - Tricky_Cooperator (score:5.0)
  - Willing@_1 (score:5.0)
  - Willing@_3 (score:5.0)
  - Willing@_4 (score:5.0)
  - Willing@_7 (score:5.0)
  - Willing@_8 (score:5.0)
  - Willing@_9 (score:5.0)
  - Worse_and_Worse@_0 (score:4.6)
  - Worse_and_Worse@_1 (score:4.68)
  - Worse_and_Worse@_2 (score:4.66)

- Worse_and_Worse@_3 (score:4.58)
- Worse_and_Worse@_4 (score:4.4)
- Worse_and_Worse@_5 (score:4.48)
- Worse_and_Worse@_6 (score:4.52)
- Worse_and_Worse@_7 (score:4.62)
- Worse_and_Worse@_8 (score:4.5)
- Worse_and_Worse@_9 (score:4.6)
- ZD-Mischief@_0 (score:1.02)

List of Figures

List of Tables

Bibliography

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

[2] Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[3] Will Knight. Alpha zeros alien chess shows the power, and the peculiarity, of ai. *MIT Technology Review*, 2017.

[4] Michael Chui. Artificial intelligence the next digital frontier? *McKinsey and Company Global Institute*, page 47, 2017.

[5] Jeffrey Horn, Nicholas Nafpliotis, and David E Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*. Ieee, 1994.

[6] Yahya Rahmat-Samii and Eric Michielssen. Electromagnetic optimization by genetic algorithms. *Microwave Journal*, 42(11):232–232, 1999.

[7] Charles Darwin and William F Bynum. *The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life*. AL Burt, 2009.

[8] The Axelrod project developers. Axelrod: v4.0.0, 2017.

[9] James Cambell. Thesis. 2016.

[10] Saul I Gass and Arjang A Assad. *An annotated timeline of operations research: An informal history*, volume 75. Springer Science & Business Media, 2005.

[11] John Tooby and Leda Cosmides. The evolution of war and its cognitive foundations. *Institute for evolutionary studies technical report*, 88(1):1–15, 1988.

[12] Robert J Aumann and Sergiu Hart. *Handbook of game theory with economic applications*, volume 2. Elsevier, 1992.

[13] Daniel M. Cable and Scott Shane. A prisoner's dilemma approach to entrepreneur-venture capitalist relationships. *Academy of Management Review*, 22(1):142 – 176, 1997.

[14] Duncan Snidal. The game theory of international politics. *World Politics*, 38(1):25–57, 1985.

[15] Bobbi S Low. *Why sex matters: A Darwinian look at human behavior*. Princeton University Press, 2015.

[16] William H Press and Freeman J Dyson. Iterated prisoners dilemma contains strategies that dominate any evolutionary opponent. *Proceedings of the National Academy of Sciences*, 109(26), 2012.

[17] Shashi Mittal and Kalyanmoy Deb. Optimal strategies of the iterated prisoner's dilemma problem for multiple conflicting objectives. *IEEE Transactions on Evolutionary Computation*, 13(3):554–565, 2009.

[18] Julie Rehmyer. Game theory suggests current climate negotiations wont avert catastrophe. *Science News*, 2012.

[19] Thomas Osang and Arundhati Nandy. Environmental regulation of polluting firms: Porter's hypothesis revisited. *Revista Brasileira de Economia de Empresas*, 3(3), 2013.

[20] Bruce Schneier. Lance armstrong and the prisoners' dilemma of doping in professional sports. *WIRED*, 2012.

[21] Jonathan Goldman and Ariel D Procaccia. Spliddit: Unleashing fair division algorithms. *ACM SIGecom Exchanges*, 13(2):41–46, 2015.

[22] spliddit.org. http://www.spliddit.org.

[23] Marc Harper, Vincent Knight, Martin Jones, Georgios Koutsovoulos, Nikoleta E Glynatsi, and Owen Campbell. Reinforcement learning produces dominant strategies for the iterated prisoners dilemma. *PloS one*, 12(12):e0188046, 2017.

[24] Daniel Ashlock, Eun Youn Kim, and Warren Kurt. Finite rationality and interpersonal complexity in repeated games. 56(2):397–410, 2004.

[25] Daniel Ashlock and Eon Youn Kim. Fingerprinting: Visualization and automatic analysis of prisoner's dilemma strategies. *IEEE Transactions on Evolutionary Computation*, 12(5):647–659, 2008.

[26] Sander Bakkes, Pieter Spronck, and Jaap Van den Herik. Rapid and reliable adaptation of video game ai. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(2):93–104, 2009.

[27] Paul C Chu and John E Beasley. A genetic algorithm for the generalised assignment problem. *Computers & Operations Research*, 24(1):17–23, 1997.

[28] Bir Bhanu, Sungkee Lee, and John Ming. Adaptive image segmentation using a genetic algorithm. *IEEE Transactions on systems, man, and cybernetics*, 25(12):1543–1567, 1995.

[29] Linus Torvald. Git.

[30] Marc, Vince Knight, Martin Jones, T.J. Gaffney, Toby Devlin, Nikoleta, and Georgios Koutsovoulos. Axelrod-python/axelrod-dojo: v0.0.8, March 2018.

[31] Seung-Seok Choi, Sung-Hyuk Cha, and Charles C Tappert. A survey of binary similarity and distance measures.